高 级 程 序 员 教 程

惠远公司内部学习资料

目 录

计算机系统结构基础知识.....9

1 计算机系统结构概述.....9

1.1 计算机系统结构、组成和实现.....9

1.2 计算机系统分类.....10

2 存储器系统.....12

1.2.1 存储器体系.....12

1.2.2 主存储器.....15

1.2.3 辅助存储器.....17

1.2.4 cache 存储器.....25

1.2.5 虚拟存储器.....29

3 流水线技术.....33

1.3.1 流水线技术的基本原理.....33

1.3.2 计算机流水线技术概述.....34

1.3.3 影响流水线性能的主要因素.....36

1.3.4 流水线计算机的存储器结构.....37

1.3.5 流水线控制.....38

1.3.6 流水线的中断处理.....39

4 输入输出系统.....39

1.4.1 输入输了控制器.....40

1.4.2 输入输出的工作方式.....42

1.4.3 设备接口.....50

1.4.4 输入输出系统的发展.....54

5 RISC 计算机.....55

1.5.4 RISC 的编译优化.....60

1.5.5 典型的 RISC 处理器.....61

6 并行处理技术.....63

1.6.1 并行处理技术概述.....63

1.6.2 并行处理机.....66

1.6.3 多处理机.....69

7 计算机安全性、可靠性及性能评价初步.....76

1.7.1 数据的安全与保密.....76

1.7.2 计算机系统的可靠性.....80

1.7.3 计算机系统的性能评价.....83

1.7.4 计算机故障诊断与容错.....87

第 1 章 计算机硬件基础知识.....90

1.1 计算机系统概述.....90

1.1.1 计算机史简介.....90

1.1.2 计算机系统的结构.....91

1.2 数据的计算机表示.....92

1.2.1 数制.....92

1.2.2 数值数据的表示方法.....96

1.2.3 符号数据的表示方法.....*103*
1.2.4 汉字的处理.....*104*
1.2.5 数据的校验方法.....*106*
1.3 计算机算术运算和逻辑运算.....113
1.3.1 二进制数的算术运算.....*113*
1.3.2 逻辑代数及逻辑运算.....*116*
1.4 计算机基本结构.....120
1.4.1 总线结构.....*120*
1.4.2 中央处理器.....*123*
1.4.3 存储器系统.....*134*
1.4.4 I／O 系统.....*141*
1.5 指令系统.....147
第 2 章 语言处理程序基础知识.....**155**
2.1 程序语言基础知识.....156
2.1.1 概述.....*156*
2.1.2 语言提供的数据类型.....*158*
2.1.3 语言所提供的控制结构.....*168*
2.2 汇编程序基本原理.....176
2.2.1 汇编语言.....*176*
2.2.2 汇编程序.....*177*
2.3 解释程序基本原理.....180
2.4 编译程序基本原理.....182
2.4.1 编译过程概述.....*182*
2.4.2 文法和语言的形式描述.....*185*
第 3 章操作系统基础知识.....**216**
3.1 操作系统概述.....216
3.1.1 什么是操作系统.....*217*
3.1.2 操作系统的功能.....*219*
3.1.3 操作系统的工作原理.....*220*
3.1.4 操作系统的硬件基础.....*221*
3.2 并发进程.....223
3.2.1 程序、进程和处理器.....*223*
3.2.2 进程间的通信.....*224*
3.2.3 信号量与P，V 操作.....*225*
3.2.4 高级通信原语.....*229*
3.2.5 管程.....*229*
3.3 系统的核心.....234
3.3.1 基本硬件机构.....*234*
3.2.2 进程.....*234*
3.3.3 进程管理.....*236*
3.4 存储管理.....237
3.4. 1 实存管理.....*239*
3.4.2 虚存组织.....*243*

3.4.3 虚存管理.....*247*
3.5 设备管理.....249
3.5.1 设备的分配.....*251*
3.5.2 磁盘调度算法.....*252*
3.6 文件管理.....253
3.6.1.文件的结构和组织.....*254*
3.6.2 文件访问方法.....*257*
3.6.3 文件存储设备管理.....*257*
3.6.4 文件控制块和文件目录.....*259*
3.6.5 文件的使用.....*261*
3.7 作业管理和用户界面.....262
3.7.1 用户作业管理.....*263*
3.7.2 作业调度.....*264*
3.7.3 用户界面.....*265*
3.8 其他管理.....265
3.8.1 死锁问题.....*265*
3.8.2 多处理器系统与线程.....*267*
3.8.3 操作系统的结构.....*268*
3.9 实例.....269
3.9.1 UNIX 系统.....*269*
3.9.2 Windows NT 系统.....*272*

软件工程.....275

第一章 软件工程概述.....275
第二章 软件需求分析.....289
第三章 软件设计.....327
第四章 程序编码.....372
第五章 软件测试.....407
第六章 面向对象的软件工程.....467
第七章 软件维护.....530
第八章 软件复用.....580
第九章 软件管理.....616
第十章 软件质量保证.....685

第 5 章 数据库基础知识.....761

5.1 关系模型.....762
5.2 关系代数.....768
5.3 关系数据库 SQL 语言.....776
5.4 关系数据库规范化理论.....800
5.5 数据库的保护.....806

第 6 章 多媒体基础知识.....815

6.1 多媒体的概念和特征.....815
6.2 图像和图形.....823
6.3 声音（音频）.....826

6.4 视频(动画) 835
6.5 多媒体数据压缩和编码技术标准 837
6.6 多媒体创作工具及发展方向 841

第 7 章 网络基础知识 844

7.1 网络的功能、分类与组成 844
7.2 网络协议和网络软件 850
7.3 构建网络与申请通信服务 860
7.4 INTERNET 和 INTRANET 初步 874
7.5 客户机/服务器模式与网络计算 879
7.6 网络的安全性 883
7.7 网络管理初步 888

第 8 章 数据结构基础 891

8.1 线性表 892
8.2 栈 901
8.3 队列 911
8.4 数组 916
8.5 字符串 928
8.6 树和二叉树 936
8.7 图 955
8.8 排序 971
8.9 查找 985

第 9 章 常用算法设计方法 995

9.1 迭代法 995
9.2 穷举搜索法 996
9.3 递推法 1000
9.4 递归 1001
9.5 回溯法 1008
9.6 贪婪法 1017
9.7 分治法 1021
9.8 动态规划法 1023

第 2 章 基本数据结构及操作 1027

2.1 数据抽象与程序设计 1027
2.2 数组 1032
2.3 队列 1066
2.4 栈 1076
2.5 集合 1088
2.6 串 1095
习题 1100

第 3 章 动态数据结构及操作 1116

3.1 链表 1116

3.2 树 1134
3.3 图 1157
习 题 1176

第 4 章 排序与查找 1188

4.1 排序 1188
4.2 查找 1215
4.3 应用举例 1226
习题 1229

第 5 章 文件与文件 I/O 操作 1243

5.1 文件的基本概念 1243
5.2 文件操作 1250
5.3 文件使用用举例 1261
习题 1268

第 6 章 事务处理流程图 1278

6.1 概 述 1278
6.2 事务处理流程 1282
6.3 事务处理流程图设计举例 1288
习题 1304

第 7 章 CASL 程序设计 1306

7.1 CASL 概述 1306
7.2 CASL 程序设计 1322
7.3 CASL 与数据结构 1350
7.4 试题分析与应试技巧 1379
习 题 1396

CASL 汇编语言程序设计(教材) 1408

第 1 章 数制及运算系统 1408
1.1 进位计数制 1409
1.2 二进制数的运算 1411
1·3 数制之间的转换 1414
1.4 数在计算机上的表示 1418
1.5 补码运算系统 1426
第 2 章 COMET 计算机及机器语言 1431
2.1 COMET 计算机结构 1432
2·2 COMET 计算机的指令结构 1435
2·3 COMET 计算机的指令系统 1441
2.4 例 题 1453
第 3 章 CASL 汇编语言及程序运行环境 1455
3.1 汇编语言 1455
3.2 伪 指 令 1457
3.3 系统宏指令 1458

3·4 第一个程序.....*1459*
3.5 CASL 运行环境.....*1467*
第 4 章 基本控制结构的程序设计.....1472
4.1 顺序型结构.....*1473*
4.2 选取型结构.....*1476*
4·3 重复型结构.....*1492*
4.5 例 题.....*1513*
第 5 章 子程序设计.....1520
5·1 子程序基本概念.....*1520*
5.4 调用中的信息传递技术.....*1522*
5.5 子程序设计应用.....*1529*
第 6 章 试题精选.....1539
6.1 控制两数操作的程序.....*1539*
6.12 数字串前导零的设置.....*1540*

试题讲解.....**1542**

1. 软件基础知识试题精解.....1542
2.1 数据结构基础.....*1542*
2.2 程序语言基础知识.....*1576*
2.3 操作系统基础知识.....*1595*
2.4 软件工程基础知识.....*1617*
2.5 数据库系统基础知识.....*1648*
2.6 多媒体基础知识.....*1669*
2. 硬件基础知识试题精解.....1676
3.1 计算机体系结构的主要部件.....*1676*
3.2 存储器系统.....*1694*
3.3 安全性、可靠性和性能评价.....*1706*
3.4 体系结构其他基础知识.....*1721*
3.5 综合性试题.....*1729*
第四章 网络基础知识试题精解.....1734
第五章 专业英语试题精解.....1757
5. 软件设计试题精解.....1777
6.1 2000 年度软件设计试题解析.....*1777*
6.2 1999 年度软件设计试题解析.....*1790*
6.3 1998 年度软件设计试题解析.....*1802*
6.4 1997 年度软件设计试题解析.....*1813*
6.5 1996 年度软件设计试题解析.....*1821*
6.6 1995 年度软件设计试题解析.....*1830*
6.7 1994 年度软件设计试题解析.....*1844*
6.8 1993 年度软件设计试题解析.....*1857*
6.9 1992 年度软件设计试题解析.....*1869*
6.10 1991 年度软件设计试题解析.....*1884*
6.11 1990 年度软件设计试题解析.....*1897*
6. CASL 语言程序编制试题精解.....1910

试题1 （2000 年试题4）.....1910
试题2 （1999 年试题4）.....1914
试题3 （1998 年试题4）.....1918
试题4 （1997 年试题4）.....1920
试题5（1996 年试题4）.....1923
试题6 （1995 年试题6）.....1926
试题7（1994 年试题6）.....1928
试题8（1993 年试题6）.....1930
试题9（1992 年试题6）.....1933
试题10（1991 年试题6）.....1936
试题11（1990 年试题7）.....1939
7．C 语言程序编制试题精解 1943
试题1（2000 年试题5）.....1943
试题2 （2000 年试题6）.....1947
试题3（1999 年试题5）.....1952
试题4（1999 年试题6）.....1956
试题5 （1998 年试题5）.....1959
试题6（1998 年试题7）.....1963
试题7（1997 年试题6）.....1966
试题8（1997 年试题8）.....1969
试题9（1996 年试题5）.....1972
试题10（1996 年试题7）.....1975
试题11（1995 年试题7）.....1978
试题12（1994 年试题7）.....1982
试题13（1993 年试题7）.....1985
试题14（1992 年试题7）.....1988
试题15 （1991 试题7）.....1990
试题16 （1990 年试题7）.....1995
历年试卷及答案（整套） 1998
1996 年试卷及答案.....1998
1997 年试题与答案.....2031
1998 年试题与答案.....2049
2000 年试题与答案.....2080
2001 年试题与答案.....2096

计算机系统结构基础知识

1 计算机系统结构概述

1.1 计算机系统结构、组成和实现

计算机系统结构（computer architecture）最初是由 G.M.Amdahl 等人在 1964 年提出的。Amdahl 在介绍 IBM360 系统时，把系统结构定义为从程序员的角度所看到的计算机的属性，即程序员要编写出能在机器上正确运行的程序所必须了解的概念性结构和功能特性。
按照计算机的层次结构，不同级别的程序员所看到的计算机属性显然不同。一般，低层机器的属性对高层机器程序员基本是透明（不可见）的，我们通常所说的计算机系统结构主要是指机器语言级机器（物理机器）的系统结构。

计算机系统结构主要研究软件、硬件功能分配，确定软件、硬件界面（机器级界面），即从机器语言程序员或编译程序设计者的角度所看到的机器物理系统的抽象。
计算机系统结构通常包括以下属性：

- 机内数据表示：硬件能直接辨识和操作的数据类型和格式；
- 寻址方式：最小可寻址单位、寻址方式的种类、地址运算；
- 寄存器组织：操作数寄存器、变址寄存器、控制寄存器及专用寄存器的定义、数量和使用规则；
- 指令系统：机器指令的操作类型、格式，指令间排序和控制机构；
- 存储系统：最小编址单位、编址方式、主存容量、最大可编址空间；
- 中断机构：中断类型、中断级别，以及中断响应方式等；
- 机器工作状态（管态、目态）的定义和切换；
- 输入输出结构：输人输出的连接方式，处理机 / 存储器与输入输出设备间的数据交换的方式，数据交换过程的控制；
- 信息保护：信息保护方式、硬件信息保护机制。

计算机组成（computer organization）指的是计算机系统结构的逻辑实现，包括机器内部数据流和控制流的组成以及逻辑设计等。其目标是合理地把各种部件、设备组成计算机，以实现特定的系统结构，同时满足所希望达到的性能价格比。一般而言，计算机组成研究的范围包括：确定数据通路的宽度，确定各种操作对功能部件的共享程度，确定专用的功能部件，确定功能部件的并行度，设计缓冲和排队策略，设计控制机构和确定采用何种可靠性技术。

计算机实现（computer implementation）是指计算机组成的物理实现。包括处理机、主存等部件的物理结构，器件的集成度和速度，器件、模块、插件、底板

的划分与连接，专用器件的设计，信号传输技术，电源，冷却及装配等技术以及相关的制造工艺和技术。
计算机系统结构、计算机组成和计算机实现是三个不同的概念，但随着新技术、新器件的出现和应用，这三者之间的界限越来越模糊了。

1.2 计算机系统分类

1.Flynn 分类法

1966 年，Michael.J.Flynn 提出根据指令流、数据流的多倍性（multiplicity）特征对计算机系统进行分类，定义如下：

指令流：机器执行的指令序列；

数据流：由指令流调用的数据序列，包括输入数据和中间结果；

多倍性：在系统性能瓶颈部件上同时处于同一执行阶段的指令或数据的最大可能个数。

Flynn 根据不同的指令流－数据流组织方式把计算机系统分成四类：

（1）单指令流单数据流（single instruction stream single data stream，SISD）

SISD 其实就是传统的顺序执行的单处理器计算机，其指令部件每次只对一条指令进行译码，并只对一个操作部件分配数据。如图 1.1（a）所示，流水方式（参见后面的 1.3 节）的单处理机有时也被当作 SISD。

（2）单指令流多数据流（single instruction stream multiple data stream，SIMD）

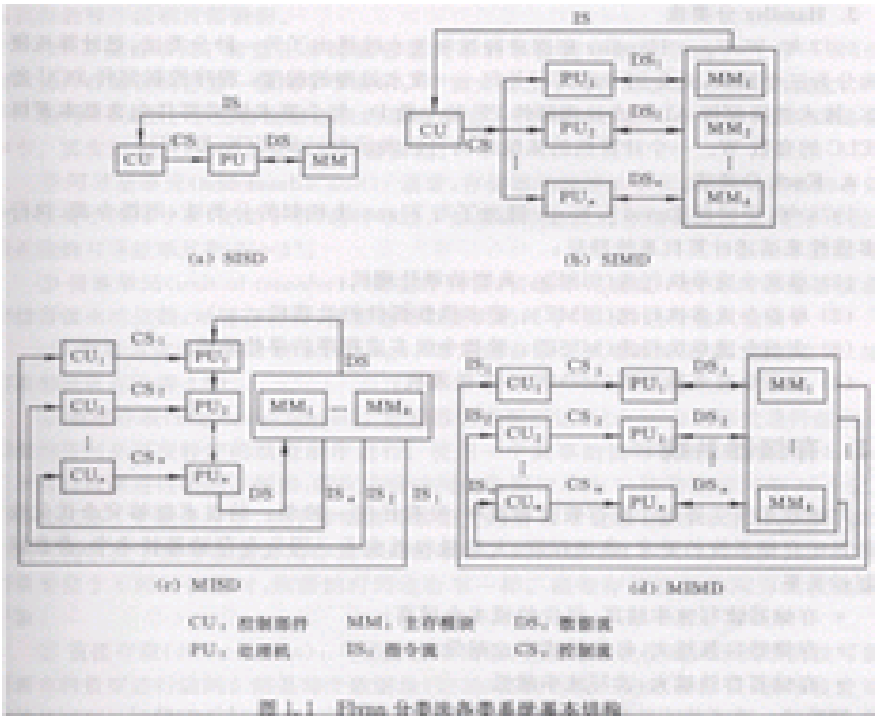
SIMD 以并行处理机（阵列处理机）为代表，其结构如图 1.1(b)所示。并行处理机包括多个重复的处理单元 PU1～PUn，由单一指令部件控制，按照同一指令流的要求为它们分配各自所需的不同数据。相联处理机也属于这一类。

（3）多指令流单数据流（multiple instruction stream single data stream，MISD）

如图 1.1(c）所示，具有 n 个处理单元，按 n 条不同指令的要求对同一数据流及其中间结果进行不同的处理。一个处理单元的输出又作为另一个处理单元的输人。这类系统实际上很少见到。有文献把流水线看作多个指令部件，称流水线计算机是 MISD。

（4）多指令流多数据流（multiple instruction stream multiple data stream，MIMD）

如图 1.1(d)所示，MIMD 是指能实现作业、任务、指令等各级全面并行的多机系统。多处理机属于 MIMD。



2.冯氏分类法

1972 年，美籍华人冯泽云教授提出用最大并行度 Pm 来对计算机系统结构进行分类。Pm 定义为计算机系统在单位时间内所能处理的最大二进制位数，平均并行度 Pa 定义为在一段时期内，计算机单位时间处理的二进制位数的平均值。

冯氏分类法把计算机系统分为四类：

（1）字串位串（word－serial and bit－serial，WSBS）：字宽=1，位宽=1；

（2）字并位串（word－parallel and bit－serial，WPBS）：字宽＞1，位宽=1；

（3）字串位并（word－serial and bit－parallel，WSBP）：字宽= 1，位宽＞1；

（4）字并位并（word－parallel and bit－parallel，WPBP）：字宽＞1 ，位宽＞1。

其中，字宽表示在一个字中同时处理的二进制位数，位宽表示在一个位片中能同时处理的字数。

3.Handler 分类法

1977 年，Wolfgan Handler 根据并行度和流水线提出了另一种分类法，把计算机硬件结构分为三个层次，并分别考虑其可并行--流水处理的程度：程序控制部件 PCU 的个数 K、算术逻辑部件 ALU 或处理部件 PE 的个数 D、每个算术逻辑部件包含基本逻辑线路 ELC 的套数 W。一个计算机的系统结构可以表示为三元组（K，D，W）。

4.Kuck 分类法

1978 年，美国的 David J.Kuck 提出了与 Flynn 法相似的分类法，用指令流、执行流和多倍性来描述计算机系统特征：

（1）单指令流单执行流（SISE）：典型的单处理机；
（2）单指令流多执行流（SIME）：带多操作部件的处理机；
（3）多指令流单执行流（MISE）：带指令级多道程序的单处理机；
（4）多指令流多执行流（MIME）：多处理机。

2 存储器系统

存储器系统主要用于保存数据和程序，没有任何一种单一的技术能够完全优化地满足计算机存储系统的要求：高速存取、大容量和低成本。因为在存储器技术中，存在以下的制约关系：

- 存储器读写速率越高，每位的成本也越高；
- 存储器容量越大，每位的成本也越低；
- 存储器容量越大，读写速率越低。

解决这一难点的方法就是采用多级存储体系结构（memory hierarchy）。

1.2.1 存储器体系

1.存储器系统的特征

（1）位置：存储器系统由分布在计算机各个不同部件的多种存储设备组成：位于 CPU 内部的寄存器，以及用于 CU 的控制存储器。内部存储器是可以被处理器直接存取的存储器，又称为主存储器（main memory），有时也简称为主存或内存；外部存储器（external memory）需要通过 I/O 模块与处理器交换数据，又称为辅助存储器（auxiliary memory），有时也简称为辅存或外存。除了正在执行的程序和处理的数据存放在主存储器外.其他所有的信息都保存在辅助存储器中，当需要时再从辅存中调人主存。

为了提高性能，弥补 CPU 处理速度和主存存取速度之间的差异，还设置了 cache（超高速缓冲存储器），其容量小但速度极快，位于 CPU 和主存储器之间，用于存放 CPU 正在执行的程序段和所需数据。

（2）存储器单元：衡量主存储器容量大小的单位通常是字节或字（1 字＝8 / 16 / 32 位），而外存储器的容量一般用字节表示。下面介绍三个与存储器单元相关的概念：

①字（word）：字是存储器组织的基本单元，一个字可以是一个字节，也可以是多个字节。其实际大小，依赖于具体的机器实现。

②可寻址单元（addressable unit）：通常，存储器的可寻址单元就是字，但有的存储器允许在更小的尺寸上（比如字节）进行寻址。如果某个系统的存储器地址宽度为 A 位，则该系统的可寻址单元数 $N=2^A$。

③传输单元（unit of transfer）：对主存而言，传输单元就是一次写人存储器或者从存储器读出的位数；对辅存而言，数据通常以更大的尺寸（如块）进行输人输出。

（3）存取方式（method of accessing）：常用的存取方式有顺序存取、直接存取、随机存取和相联存取等 4 种。

①顺序存取（sequential access）：存储器的数据以记录（record）的形式进行组织。对数据的访问必须按特定的线性顺序进行。使用一个共享的读写装置（R / W mechanism）对所有的数据进行访问。例如，假定当前访问的数据记录为 1，接着要访问第 N 个记录，

就需要从第 1 个记录开始，顺序移动读写装置直到第 N 个记录，然后才能对它进行读写。访问时间取决于读写装置访问存储单元的顺序。同样是对第 N 个记录进行读写，当读写装置位于不同的位置时，所需的时间也会不一样。磁带存储器的访问方式就是顺序存取。

②直接存取（direct access）：与顺序存取相似，直接存取也使用一个共享的读写装置对所有的数据进行访问。但是每个数据块（记录）都拥有惟一的地址标识，读写装置可以直接移动到目的数据块的所在位置进行访问。存取时间也是可变的。磁盘存储器就采用直接存取的方式。

③随机存取（random access）：存储器的每一个可寻址单元都具有自己惟一的地址和读写装置，系统可以在相同的时间内对任意一个存储单元的数据进行访问，而与先前的访问序列无关。主存储器即是一例。

④相联存取（associative）：这也是一种随机存取的形式，但是选择某一单元进行读写是取决于其内容而不是其地址。与普通的随机存取方式一样，每个单元都有自己的读写装置，读写时间也是一个常数。使用相联存取方式，可以对所有

的存储单元的特定位进行比较，选择符合条件的单元进行访问。为了提高地址映射的速度，cache 可能采取该方式讲行访问。

（4）性能：存储器系统的性能主要由存取时间、存储器带宽、存储器周期和数据传输率等来衡量。

①存取时间（access time）：对于随机存取而言，存取时间就等于完成一次读或写所花的时间；对于非随机存取而言，存取时间就等于把读写装置移动到目的位置所需的时间。

②存储器带宽（bandwidth）：每秒钟能访问的位数。如果存储器周期是 500ns，而每个周期可访问 4 字节，则带宽为 64Mbps。

③存储器周期（memory cycle time）：主要是针对随机存取而言，一个存储器周期就等于两次相邻的存取之间所需的时间。

④数据传输率（transfer rate）：每秒钟输入 / 输出的数据位数。对 TN=T 于随机存取而言，数据传输率 R=1 / 存储器周期；对于非随机存取而言，$T_N= T_A+N/R$，T_N：读写 N 位所需的平均时间，T_A：平均存取时间，R：数据传输率（bps），则数据传输率 $R=N/（T_N－T_A）$

（5）物理介质：常用的存储介质有半导体存储器（semiconductor memory）、磁介质存储器（magnetic surface memory）、光存储器三种。

（6）物理特性

①易失性存储器（volatile memory）：当断电时，保存在易失性存储器里的内容就会丢失。

②非易失性存储器（nonvolatile memory）：保存在非易失性存储器里的内容不会因为断电而丢失。除非对其进行写入操作，否则数据不会被破坏。

③不可改写存储器（nonerasable memory）：数据一旦写入，就无法更改。

2.多级存储器体系

计算机采用多级存储器体系的目的就在于能够获得尽可能高的存取速率，同时保持较低的成本。
如图 1.2 所示，从存储器体系顶端的寄存器和 cache 到底部的磁带存储器，其价格逐渐降低；而容量和访问时间则逐渐增加。通过合理地设计存储器体系结构，使得经常访问的数据保存在高速存储器中，而较少访问的数据保存在低速存储器中，即从高到低，数据的访问频率逐渐减少。这样就可能组成一个访问速度尽可能快，容量尽可能大而价格也较便宜的存储系统。



图 1.2 计算机的存储体系

1.2.2 主存储器

1.主存储器的种类

（1）RAM（random access memory）：RAM 存储器既可以写入也可以读出，但断电后信息无法保存，因此只能用于暂存数据。RAM 又可分为 DRAM 和 SRAM 两种：

①DRAM（dynamic RAM）：信息会随时间逐渐消失，因此需要定时对其进行刷新（refresh）维持信息不丢失。

②SRAM（static RAM）：在不断电的情况下信息能够一直保持而不会丢失。DRAM 的密度大于 SRAM 且更加便宜，但 SRAM 速度快，电路简单（勿需刷新电路），然而容量小，价格高。

（2）ROM（read only memory）：只读存储器，信息已固化在存储器中。ROM 出厂时其内容由厂家用掩膜技术（mask）写好，只可读出，但无法改写。一般用于存放系统程序 BIOS 和用于微程序控制。

（3）PROM（programmable ROM）：可编程 ROM，只能进行一次写入操作（与 ROM 相同），但是可以在出厂后，由用户使用特殊电子设备进行写入。

（4）EPROM（erasable PROM）：可擦除的 PROM，其中的内容既可以读出，也可以写入。但是在一次写操作之前必须用紫外线照射 15～20 分钟以擦去所有信息，然后再写入。可以写多次。

（5）E^2PROM（electrically EPROM）：电可擦除 EPROM，与 EPROM 相似，可以读出也可写入，而且在写操作之前，不需要把以前内容先擦去。能够直接对寻址的字节或块进行修改，只不过写操作所需的时间远远大于读操作所需时间（每字节需几百 ms），其集成度也较低。

（6）闪速存储器（flash memory）：其性能介于 EPROM 与 E^2PROM 之间。与 E^2PROM 相似，可使用电信号进行删除操作。整块闪速存储器可以在数秒内删除，速度远快于 EPROM；而且可以选择删除某一块而非整块芯片的内容，但还不能进行字节

级别的删除操作。集成度与 EPROM 相当，高于 E^2PROM。闪速存储器有时也简称为闪存。

2.主存储器的组成

（1）存储器单元（memory cell）：一个简单的存储器单元如图 1.3 所示。



图 1.3 存储单元示意图

①存储器单元的基本性质：具有两个稳定状态用以表示"1"和"0"；至少可以写入一次；可以读出（tosense the state）。

②存储器单元的操作：select 信号用于选择该单元进行读写；control 信息用于表明执行读还是写操作；data 线用于数据的输入输出。

（2）存储器芯片（memory chip）：由多个单元矩阵组成芯片，再由若干芯片组成存储体。现代计算机一般都采用半导体存储器。近来半导体工艺发展很快，以 DRAM 为例，集成度平均每年提高 60%，但价格却不断下降。

我们以 DRAM 芯片为例，讲解存储器工作的基本原理，如图 1.4 所示。

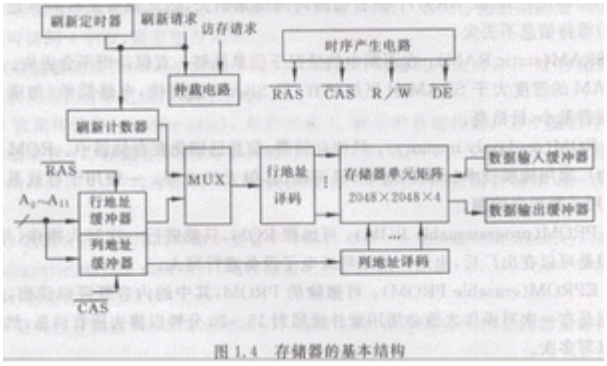


图 1.4 存储器的基本结构

① 存储单元矩阵：由 4 个 2048×2048 单元矩阵组成存储体。

② 地址信号：寻址 2048×2048 个字需要 22 条地址线（2^{22}=2048×2048），但为了节省芯片的引脚，只使用 A_0～A_{10}11 条地址线。由行选信号 $\overline{RAS}$ 和列选信号 $\overline{CAS}$ 分别控制。用 $\overline{RAS}$ 信号控制 A_0～A_{11} 从 2048 行中选出一行，用 $\overline{CAS}$ 信号控制 A_0～A_{11} 再从 2048 列中选出一列。

③ 时序发生器：在时钟信号与控制信号的控制下，产生存储器读写所需的时序

信号： $\overline{RAS}$、 $\overline{CAS}$、 $\overline{W}$和 $\overline{DE}$信号。

④ 刷新电器：由刷新定时器定时产生刷新请求，启动刷新计数器产生刷新所需的行地址。因为刷新时是将该行所有存储单元进行刷新，因此只需提供刷新行地址，每发出一次刷新请求，计数器值加 1，以便刷新下一行。

假定 2ms 对全部 DRAM 刷新一次,而共有 2^{11} =2K 行,因此约 1μs(2ms/2K≈1μs),即需发出一次刷新请求。

⑤ 仲裁电路：当同时存在访存请求和刷新请求时，由仲裁电路依据机器的状态判断优先处理哪个请求。

（3）存储器的组成：实际的存储器总是由一片或多片存储芯片配以控制电路组成的。其容量往往用 W×B 来表示。W 表示该存储器的存储单元（word）的数量而 B 表示每一个 word 由多少 bit 组成。如果某一芯片规格为 ω× b，则用其组成 W ×B 的存储器需要用(W/ω)×(B/b)个芯片。
首先是 B/b 个芯片一组，每组芯片共有 ω 个字，每个字则有 B 个位，一共用 W / ω 个组来构造所需的存储器。

①分组：如图 1.5 所示，B/b 个芯片组成一组，共用地址线、读写线 R／W 和片选线
CS；它们的数据线组成了 B 位宽的数据。

②组间互联：如图 1.6 所示，用高位地址线译码产生组选择信号 CS，低位地址线则用来在组内寻址。

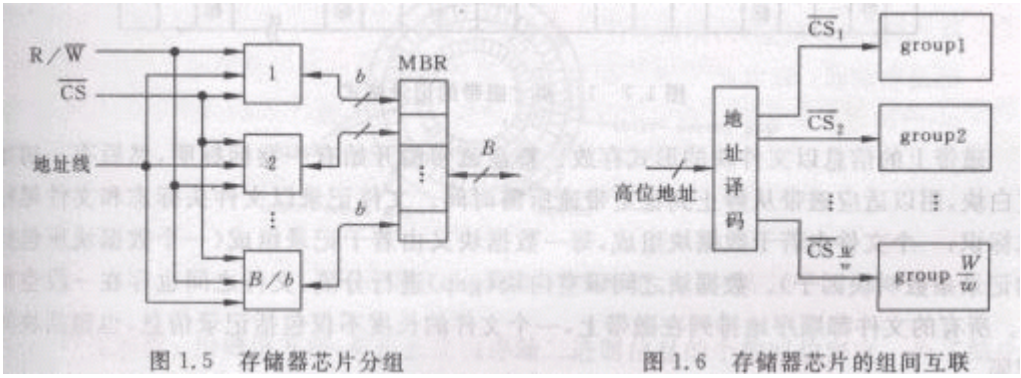


图 1.5 存储器芯片分组

图 1.6 存储器芯片的组间互联

1.2.3 辅助存储器

辅助存储器用于存放当前不需要立即使用的信息，一旦需要，再和主机成批交换数据，是主存储器的后备，因此称之为辅助存储器；它又是主机的外部设备，又称之为"外存储器"。辅助存储器的最大特点是存储器容量大、可靠性高、价格低。主要可分为磁表面存储器和光存储器两大类。

1.磁表面存储器

磁表面存储器目前主要有磁带和磁盘存储器。

（1）磁记录原理：磁表面存储器通过磁头来进行读写。磁头线圈中通入一定方向和大小的脉冲电流，磁化磁头的导磁体，从而建立起一定方向和强度的磁场。当磁介质从磁头下经过时，从磁头间隙处将其磁化。由于剩磁效应的存在，当磁场消失后，磁介质表面仍存在剩磁。线圈中电流方向不同，磁介质上被磁化的方向不同--用以代表"1"和"0"。随着电流的变化以及磁介质磁头间的相对运动，就可以把二进制信息序列转化为介质表面的磁化单元序列。读出过程则是逆过程，即把记录的磁化单元序列还原成电流脉冲序列。

（2）磁带存储器：磁带存储设备是一种顺序存取的设备，存取时间较长，但存储容量大，便于携带，价格便宜，所以也是一种主要的辅助存储器。磁带的内容由磁带机进行读写（最便宜也最慢）。

①工作原理：按磁带机的读写方式主要可以分为两种，启停式和数据流。

• 启停式磁带机：磁带按带宽可以分为 1 / 4 英寸、1 / 2 英寸和 1 英寸三种。以 1 / 2 寸为例，其标准格式如图 1.7 所示。

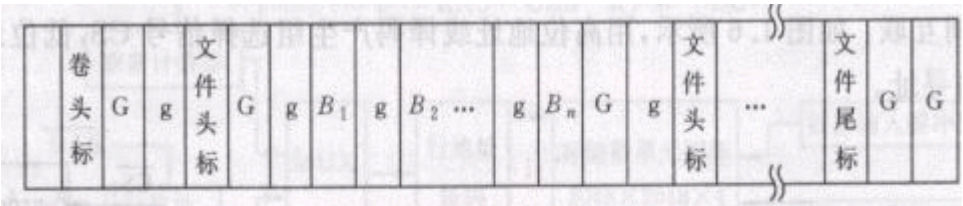


图 1.7 1/2 英寸磁带的记录格式

磁带上的信息以文件块的形式存放。整盘磁带的开始有一卷标标明，然后有一初始空白块，用以适应磁带从静止到稳定带速所需时间。文件记录以文件头标志和文件尾标志标识，一个文件由若干数据块组成，每一数据块又由若干记录组成（一个数据块所包括的记录条数叫块因子）。数据块之间以空白块（gap）进行分隔，文件之间也存在一段空隙 G。所有的文件都顺序地排列在磁带上，一个文件的长度不仅包括记录信息，也包括块间间隔。

磁带机每一次读写信息的位数与磁带表面并行记录信息的磁道数有关：如 7 道、9 道和 16 道，则分别有 7，9，16 个磁头并列，一次可以读写 7 位、9 位或 16 位。

• 数据流磁带机：数据流磁带机结构简单，价格低，数据传输速率快。数据流磁带机的数据格式见图 1.8。其记录格式是串行逐道记录信息，每次读写 1 位信息，数据连续地写在磁带上，数据块之间以空隙分隔。磁带机不能在块间启停。读写

顺序如下：（4 个磁道）先从 0 道的首端（BOT）开始，到其末端（EOT）；然后第 1 道反向记录从 EOT 到 BOT，而 2 道又正向从 BOT 到 EOT，3 道再反向。

②性能参数

• 数据传输速率（B / s）=磁带记录密度（B / mm）×带速（mm/s）

• 数据块的长度= B_1（记录数据所需的长度）＋ B_2（块间间隔）

• B_1=（字节数/记录）×块因子/记录密度

• 读 N 条记录所需的时间：T=S（启停时间）＋R＋D

R（有效时间）=（N×字节数 / 记录）/ 传输速率

D（间隔时间）=块间隔总长 / 带速=［（N / 块化因子）×（块间间隔）］/ 带速

（3）磁盘存储器

①数据的组织：如图 1.9 所示，磁盘上的数据都存放于磁道上。磁道就是磁盘上的一组同心圆，其宽度与磁头的宽度相同。为了避免减少干扰，磁道与磁道之间要保持一定的间隔（inter－track gap），沿磁盘半径方向，单位长度内磁道的数目称之为道密度（道 / 英寸，TPI），最外层为 0 道。

• 位密度：沿磁道方向，单位长度内存储二进制信息的个数叫位密度。为了简化电路设计，每个磁道存储的位数都是相同的，所以其位密度也随着从外向内而增加。

• 扇区：磁盘的数据传输是以块为单位的，所以磁盘上的数据也以块的形式进行存放。这些块就称为扇区（sector），每个磁道通常包括 10～100 个扇区。同样为了避免干扰，扇区之间也相互留有空隙（inter－sector gap）。

• 柱面：柱面是若干个磁盘组成的磁盘组，所有盘面上相同位置的磁道组称为一个柱面（每个柱面有 n 个磁道）；若每个磁盘有 m 个磁道，则该磁盘组共有 m 个柱面。

②数据记录格式：为了让磁头读写定位机构能够确定磁盘上某一磁道的具体位置，因此在新磁盘使用之前，必须对其进行格式化，以写入控制数据供磁盘驱动器使用，该数据对用户不可见。从没有格式化过的盘叫做白盘，上面只是记录了有多少磁道、磁道密度等全球统一标准，还不能直接使用。Seagate ST506 的数据格式如图 1.10 所示。

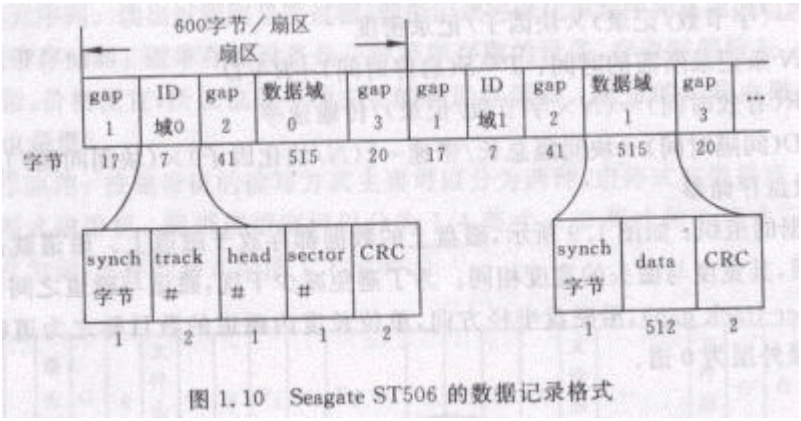


图 1.10 Seagate ST506 的数据记录格式

③磁盘存储器的特征

• 磁头：固定磁头磁盘（fixed－head disk）：磁头不沿半径方向运动，每一磁道都有一个用于读写数据的磁头。可移动头磁盘（movable－head disk）：每一面只有一个磁头，可以沿径向运动，停留在任一磁道的上方。

• 磁盘驱动器：磁盘必须置于驱动器才能进行读写。不可替换盘（nonremovable disk 的盘片永久固定在驱动器内不可更换。可替换盘（removable disk）则可以替换驱动器内的盘片，因此可以在不同系统中交换盘片使用。

面：单面 / 双面磁盘。
磁盘组：由多个磁盘组成。
磁头机构：根据磁头与磁盘之间的间隔可以分为三类：

• 接触式（contact）：磁头在读写时与盘面接触，用于软磁盘（floppy disk）存储器。

• 固定空隙（fixed gap）：留有一个固定的空气隙（air gap）、为了对磁盘进行读写，磁头必须感应或产生相当强度的电磁场。为了提高数据密度就需要提高道密度、即缩小道的宽度--也就是需要减少磁头的尺寸，这样一来，磁头就必须更加接近盘面以满足读写需求。但是这样由于灰尘杂质的影响更加容易出错。

• 温盘(winchester)：磁盘置于密封的驱动器内以避免灰尘杂质的影响，磁头更加接近盘面以获取更高的数据密度。当磁盘静止时磁头轻置于盘面上，当磁盘旋转时其产生的空气压力足以使磁头悬浮起来。因此可以使用更小尺寸的磁头。

硬盘的存储介质是铝合金等"硬"材料，并在其表面覆以磁性材料。软盘的介质是以塑料等软材料制成，其成本低、重量轻，易携带易保存，给计算机信息的转储、拷贝提供了极大的方便；但速度慢，易损坏，存储容量小。

④性能参数

·存储容量（capacity）：磁盘格式化后能够存储有用信息的总量。存储容量=n×t×s×b，其中：n 为保存数据的总盘面数；t 为每面磁道数；s 为每道的扇区数；b 为每个扇区存储的字节数。

·存取时间（access time）：包括寻道时间和等待时间。寻道时间（查找时间，seek time）为磁头移动到目标磁道所需的时间（movable－head disk），对于固定磁头磁盘而言，无需移动磁头，只需选择目标磁道对应的磁头即可。等待时间为等待读写的扇区旋转到磁头下方所用的时间。一般选用磁道旋转一周所用时间的一半作为平均等待时间。寻道时间由磁盘机的性能决定，目前主流硬盘典型的 AST（average seek time）一般在 10ms 左右，而转速则有 2400rpm，5400rpm，7200rpm 等等。软盘转速较慢，一般只有 360rpm（因为磁头与盘面接触性读写）。

·数据传输速率：磁头找到地址后，单位时间写入或读出的字节数。R=TB／T，其中：TB 为一个磁道上记录的字节数，T 为磁盘每转一圈所需的时间，R 为数据传输速率。

（4）RAID 存储器：因为辅助存储器性能改进的速度远远跟不上处理机以及主存储器性能的改进，所以存储系统性能的提高也就是改善整个计算机系统性能的关键。并行系统结构提供了一种采用低成本器件技术获得高性能的方法。

1988 年美国加州大学伯克利分校 D.A.Patterson 教授提出了廉价磁盘冗余阵列（redundant array of inexpensive disks，RAID）技术，旨在缩小日益扩大的 CPU 速度和磁盘存储器速度之间的差距。其策略是用多个较小的磁盘驱动器替换单一的大容量磁盘驱动器，同时合理地在多个磁盘上分布存放数据以支持同时从多个磁盘进行读写，从而改善了系统的 I／O 性能。小容量驱动器阵列与大容量驱动器相比，具有成本低，功耗小，性能好等优势；低代价的编码容错方案在保持阵列的速度与容量优势的同时保证了极高的可靠性。同时也较容易扩展容量。但是由于允许多个磁头同时进行操作以提高 I／O 数据传输速度，因此不可避免地提高了出错的概率。为了补偿可靠性方面的损失，RAID 使用存储的校验信息（stored parity information）来从错误中恢复数据。最初，inexpensive 一词主要针对当时另一种技术（single large expensive disk，SLED）而言，但随着技术的发展，SLED 已是明日黄花，RAID 和 non－RAID 皆采用了类似的磁盘技术。因此 RAID 现在代表独立磁盘冗余阵列（redundant array of independent disks），用 independent 来强调 RAID 技术所带来的性能改善和更高的可靠性。

RAID 机制中共分 6 个级别，工业界公认的标准分别为 RAID0～RAID5，其共同特征如下：

- RAID 由若干个物理驱动器组成，但对操作系统而言仍是一个逻辑驱动器；
- 数据分布在阵列中多个驱动器上；
- 冗余的磁盘容量用以保存奇偶信息（parity），以便在磁盘失效时进行恢复（RAID0 不支持该特征）。

RAID 应用的主要技术有：

• 分块技术（striping）：对主机请求读写的数据进行分块，使之分布于多台磁盘上；

• 交叉技术（interleaving）：对存放在多台磁盘上数据的读写采取交叉方式进行；

• 重聚技术（declustering）：对多台磁盘上的存储空间进行重新编址，使数据按重新编址后的空间进行存放。

①RAID0 级（无冗余和无校验的数据分块）：具有最高的 I / O 性能和最高的磁盘空间利用率，易管理，但系统的故障率高，属于非冗余系统，主要应用于那些关注性能、容量和价格而不是可靠性的应用程序。

所有的系统 / 用户数据表面上看起来存放在一个逻辑驱动器上，但实际上这个驱动器被分成若干块（strip，可能是 physical blocks，sectors…），这些 strip 轮转地映射到相邻的阵列磁盘中。

②RAID1 级（磁盘镜像阵列）：由磁盘对组成，每一个工作盘都有其对应的镜像盘，上面保存着与工作盘完全相同的数据拷贝，具有最高的安全性，但磁盘空间利用率只有 50%。RAID1 主要用于存放系统软件、数据以及其他重要文件。它提供了数据的实时备份，一旦发生故障所有的关键数据即刻就可使用。

RAID1 通过简单的数据备份实现冗余，也使用了数据分块技术。任一个逻辑块都对应着两个（映射到）不同的物理驱动器上。对块的读请求可以由两个盘中任一个进行服务，往往是由寻址时间短的驱动器进行服务。对块的写请求，两个块都需要更新，但是可以并行进行。因为无需计算校验位，所以比 RAID2～5 的写入更快。很容易从失效中恢复。但缺点是代价很大。

③RAID2 级（采用纠错海明码的磁盘阵列）：采用了海明码纠错技术，用户需增加校验盘来提供单纠错和双验错功能。对数据的访问涉及到阵列中的每一个盘。大量数据传输时 I/O 性能较高，但不利于小批量数据传输。实际应用中很少使用。

④RAID3 和 RAID4 级（采用奇偶校验码的磁盘阵列）：把奇偶校验码存放在一个独立的校验盘上。如果有一个盘失效，其上的数据可以通过对其他盘上的数据进行异或运算得到。读数据很快，但因为写入数据时要计算校验位，速度较慢。

RAID2 和 RAID3 使用了并行存取技术（parallel access）。对每一个 I / O 请求，所有的盘都要参与响应。RAID2 和 RAID3 也使用了数据分块技术，但是块非常的小，通常是 1 个字节或 1 个字。因为数据以很小的块进行组织，RAID3 可以达到很高的数据传输率。任一个 I / O 请求都可能会引起多个磁盘的并行数据传输，尤其是大量数据时，性能更优异。但另一方面，一次 RAID3 只能处理一个 I/O

请求，所以不适合有大量 I/O 请求的应用。

在 RAID4 中，有专门一个校验盘（parity disk），用于存放校验块。对于各数据盘上的相应块，逐位进行奇偶运算求出其校验码，并存放在校验盘相应的块上。

当对 RAID4 进行少量数据的写入时，性能会有明显下降，因为阵列管理系统不仅需要更新用户数据而且同时还要重新计算奇偶校验位，每一块的写入而要两次读和写操作。

若是大量数据的定入，对所有盘上的块进行写入，奇偶位的校验就只需对新数据进行计算，勿需额外读写。

⑤RAID5（无独立校验盘的奇偶校验码磁盘阵列）：与 RAID4 类似，但没有独立的校验盘，校验信息分布在组内所有盘上，对于大、小批量数据读写性能都很好。RAID4 和 RAID5 使用了独立存取（independent access）技术，阵列中每一个磁盘都相互独立地操作，所以 I/O 请求可以并行处理。所以，该技术非常适合于 I/O 请求率高的应用而不太适应于要求高数据传输率的应用。与其他方案类似，RAID4、RAID5 也应用了数据分块技术，但块的尺寸相对大一点。

RAID 一般采用自带微处理器的控制卡，由其完成阵列的控制、数据的分块与组合、并行高度等功能。RAID 保证了存储系统的可靠性和连续工作能力，同时其并行处理能力也显著提高了 I/O 性能。RAID 也具有海量存储能力。

2.光存储器

自从 1983 年 CD（compact disk）数字音频系统大获商业成功以来，极大地推动了发展低成本的光存储器技术作为计算机系统辅助存储器。一系列的光盘（optical disk)产品纷纷推出：CD，CD-ROM，CD-I，DVI，WORM 以及 EOD(erasable optical disk)。

（1）光盘的存取原理：光盘存储器是利用激光束在记录表面存储信息，根据激光束的反射光来读出信息。

（2）CD-ROM：CD-ROM 是在塑胶圆盘（polycarbonate）的表面覆以一层高反光物质--通常是铝（aluminum）制成的。数字化编码信息是以其反光表面一系列极其微小的凹坑来记录的。

①制作过程

·首先制作母盘（master）：用聚焦极细（直径不超过 1μm）的高强度激光束照射到介质上，使其局部加热熔化形成一个极微小的凹坑，从而改变其光学特性，以此来记录信息；

·再用该母盘制造一个模子用来复制拷贝；

·在记录表面（有凹坑）覆盖一层透明的清漆进行保护，防止灰尘或擦伤。

②读出原理：CD-ROM 的读出是通过用低功率的激光束来照射盘面，有凹坑和无凹坑处的反射光强度不相同，光电检测电路把这些再转换为数字信号即可读出信息。

③读取技术：CD-ROM 的读取目前有三种方式：

·恒定角速度（constant angular velocity,CAV）:光驱主轴转速保持恒定，因而光盘内外圈的线速率不一致，为了保持其读盘速率一致，这样就需改变内外圈的记录密度。其优势在于：由于主轴转速保持恒定，所以可以直接寻址，存取时间较短；不足之处在于浪费了存储空间，所以很少在 CD-ROM 上使用。

·恒定线速度（constant linear velocity，CLV）：光驱主轴转速随读取的位置不同而变化，以保证光盘内、外圈的线速率一致。这样激光头适应性较高，读盘率也较高。但在增加存储量的同时，也增加了寻址时间。因为主轴速率必须不停变化，因而光驱转速不能太高。

·部分恒定角速度（partial－CAV，PCAV）：CLV、CAV 都有各自的缺陷，PCAV 对 CAV 方式进行了改进，以自适应转速的方式来尽可能维持最高角速度。一旦由于线速率过高而无法正确读出信号，就会自动降低转速，直到从光盘反射回的信号足够清晰。只要可能，光驱总是以全速旋转。这样在大容量和高速度之间取得折衷。

④CD-ROM 优缺点：CD-ROM 非常适用于把大批量数据分发给大量的用户。与传统磁盘存储器相比，有以下优点：具有更大的容量，可靠性高，光盘的复制更简易，可更换，便于携带；其缺点是：只读，存取时间比较长。

⑤CD-ROM 驱动器：CD-ROM 驱动器从最初的单倍速（150KB/s）已经发展到目前的 48 倍速（7200KB / s），甚至更高的 CD-ROM 驱动器也已问世。

（3）WORM（write－once read－many）：WORM 盘允许用户使用一种较 CD-ROM 昂贵的驱动器对其写入数据，一旦写过之后，即不能再进行写操作但可以随意读。为了提供较快的存取速率，WORM 使用了 CAV 技术，以牺牲容量为代价。

WORM 的制作过程如下：先用高功率的激光束照射光盘，使之在表面产生一系列的泡（blister），即可提供给用户使用。WORM 的写入过程：当已经预格式化的盘放入到 WORM 驱动器后，使用功率较低的激光束照射光盘，其强度刚好使气泡破裂（burst blitsters）。通过使气泡破裂或不破裂就可以记录信息。WORM 的读出也很简单，用激光束照射盘面，很容易识别出气泡破裂处，使用简单的电路即可完成读操作。

（4）EOD：EOD 盘可以反复地进行读写，如同普通的磁盘。最常用的是磁光盘，利用磁光效应来进行读写。其写入过程，用激光束来加热介质，从而改变（磁光

材料介质）其局部磁场的方向来记录信息。由于该极化过程（polarization process）并不会引起光盘物理特性的改变，所以该过程可以反复进行。其数据的读出也是通过磁光效应来进行的。

与 WORM 类似，EOD 也使用 CAV 方式读取。

1.2.4 cache 存储器

设置 cache（高速缓冲存储器）的目的在于提高 CPU 数据输入输出的速率，突破所谓的"冯·诺依曼瓶颈"，即 CPU 与存储系统间数据传送带宽限制。高速存储器能以极高的速率进行数据的访问，但因其价格高昂，如果计算机的主存储器完全由这种高速存储器组成则会大大增加计算机的成本。通常在 CPU 和主存储器之间设置小容量的高速存储器 cache，如图 1.11 所示。cache 容量小但速度快，主存储器速度较低但容量大，通过优化调度算法，系统的性能会大大改善，仿佛其存储系统容量与主存相当而访问速度近似 cache。在计算机的存储系统体系中，cache 是访问速度最快的层次。

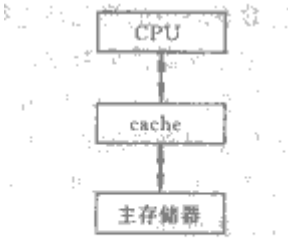


图 1.11 cache/主存储器两级体系结构

1.cache 存储器的原理

使用 cache 改善系统性能的依据是程序的局部性原理，即程序的地址访问流有很强的时序相关性，未来的访问模式与最近已发生的访问模式相似。研究结果表明，在任一给定的时间间隔内，对不同的地址区域其访问概率是不同的，有的区域高，有的区域低。而另一种可能则是访问概率随着离当前执行指令的远近而变化，离当前执行指令越近，其概率也越高。

依据局部性原理，把主存储器中访问概率高的内容存放在 cache 中，当 CPU 需要读取数据时就首先在 cache 中查找是否有所需内容，如果有则直接从 cache 中读取；若没有再从主存中读取该数据，然后同时送往 CPU 和 cache。如果 CPU 需要访问的内容大多都能在 cache 中找到（称为访问命中，hit），则可以大大提高系统性能。

如果以 h 代表对 cache 的访问命中率（hit rate），t_1 表示 cache 的周期时间，

t_2表示主存储器周期时间，以读操作为例，使用"cache 十主存储器"的系统的平均周期为t_3 则：

t_3 =h × t_1 +（1 一 h）× t_2

其中，（1-h）又称为失效率（未命中率，miss rate）。假设 cache 的读写速度是主存的 10 倍，即t_1 =0.1t_2 ，当访问命中率为 0.99 时：

t_3 =0.99 ×0.1 t_2 +0.01 t_2 = 0.109 t_2

可见当 cache 的命中率很高时，系统的存储器访问速度高于主存而已经接近 cache。
当访问命中率 h=0.90 时：

t_3 =0.90 ×0.1 t_2 +0. 1 t_2 = 0.1 9 t_2

系统的平均存储周期与命中率有很密切的关系，命中率的提高即使很小也能导致性能上的较大改善。

2.cache 存储器的结构

cache 存储器的组织结构与主存储器不一样，它以行（line）作为基本单元。每一行又分为标志项和数据域两部分。数据域中存放着若干项数据，而标志项则是这一块数据的地址标识。

当 CPU 发出对存储器的读命令后，其访问地址先送给 cache 控制器，。ache 检查其地址标识符目录以确定是否有匹配项。若发现匹配项（命中），则根据其访问地址确定是对该行数据块中的第几项进行读取，然后该项即进入 cache 的数据寄存器。如果没有命中，则去主存储器读取数据。这时不仅仅只是读该地址指定的存储单元而且把其相邻的 K 一 1 个单元(K 是 cache 中一行所能保存的数据单元个数)的内容都读入 cache 中保存。

因为根据局部性原理，这一块数据很有可能将被 CPU 访问。同时把指定的存储单元的内容送入 CPU。如果 cache 存储器已满，还要根据某种淘汰算法从 cache 中清除一行以存放该数据块。

3.cache 存储器的映射机制

当 CPU 发出访存请求后，存储器地址先被送到。ache 控制器以确定所需数据是否已在 cache 中，若命中（hit）则直接对 cache 进行访问。这个过程称为 cache 的地址映射（mapping）。为了适应 cache 存储器的极高存取速率，映射也必须在极短的时间内完成。常见的映射方法有直接映射、相联映射和组相联映射。

(1) 直接映射（direct mapping）

这种方法以随机存取存储器作为 cache 存储器，硬件电路较简单，其组织结构如图 1.12。

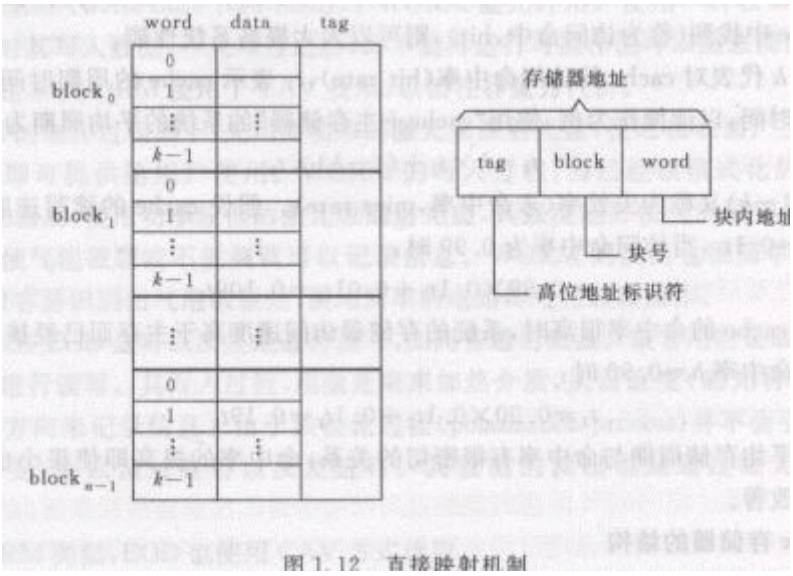


图 1.12 直接映射机制

cache 每行以块（block）形式组织：图中 cache 共有 n 个块，由其块号来标识；其中每一块又包含 k 个数据项，每一数据项由一个指针（word）标识。每块中所有的数据项具有相同的地址高位标识符（tag）。

在进行映射时，存储器的地址被分成三部分，从高到低依次为：高位地址标识符（tag）、块号（block）以及块内地址（word）。首先按照块号访问 cache，把该块的 tag 值与存储器地址中的 tag 域进行比较，若相同则根据地址中的 word 域对该项数据进行读写，若 tag 不相符则说明访问未命中。

设存储器地址长度为 1，则 word 域的长度为 $\log_2 k$，而 block 域长度为 $\log_2 n$，地址的高 $l-\log_2 n-\log_2 k$ 位用作 tag 域。

（2）相联映射（associative mapping）

这种方法使用相联存储器（associative memory）作为 cache，其速度快于直接映射，但是硬件电路较复杂，而且价格也较昂贵。

相联存储器是一种特殊的存储器，和常见的 RAM（random access memory）不同，它是一种基于数据内容进行访问的存储设备 CAM（content addressable memory）。当对其写入数据时，相联存储器能够自动选择一个未用的空单元进行存储；当要读出数据时，不是给出其存储单元的地址，而是直接给出该数据或者该数据的一部分内容，相联存储器对所有的存储单元中的数据同时进行比较并标记符合条件的所有数据以供读取。由于比较是同时、并行进行的，所以这种基于数据内容进行读写的机制，其速度比基于地址进行读写的方式要快许多。

使用相联存储器组成的 cache 存储器，其基本单元分成两部分：地址部分和数据部分。数据部分用于存放数据，而地址部分则存放该数据的存储器地址。当进行映射时，相联存储器把 CPU 发出的存储器地址与 cache 内所有的地址信息同时进行比较，以确定是否命中。

（3）组相联映射（set－associative mapping）

从直接映射机制可以发现，具有相同的 block 与 word 号但 tag 不同的数据（即地址低位相同而高位不同）不能同时存放在 cache 中。它的一种改进型就是组相联映射机制，该机制下。ache 存储器的组织结构如图 1.13 所示。

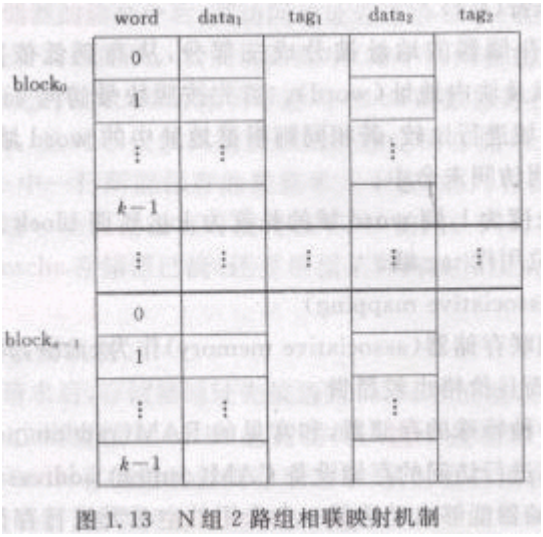


图 1.13 N 组 2 路组相联映射机制

组相联映射与直接映射的不同之处在于：每一个块中有两个或两个以上不同的 tag 域，每一个 tag 域则对应着一个数据域。这样。ache 中同一项的两个（或多个）数据有相同的 block 值和 word 值，但是可以有不同的 tag 值，从而弥补了直接映射机制的缺陷。

当 CPU 发出读写请求后，地址信息中的 block 域用来访问 cache 存储器。存储器地址所含的 tag 域则同时和该块中所有的 tag（图中为两个）进行比较以确定是否命中。为了加快 tag 的比较过程，可以采用相联存储器的方式并行地进行比较。这就是组相联名称的由来。

4.cache 存储器的淘汰算法

当 cache 存储器产生了一次访问未命中之后，相应的数据应同时读入 CPU 和 cache。但是当 cache 已存满数据后，新数据必须淘汰。ache 中的某些旧数据。最常用的淘汰算法有随机淘汰法、先进先出法（FIFO）和近期最少使用淘汰法（least recently used，LRU）。

（1）随机淘汰法
从 cache 的各行中随机选取一行进行淘汰。

（2）FIFO 法

cache 控制器设置特殊的时间标志以记录数据在 cache 中存放的时间，当需要淘汰时，则选择进入 cache 时间最长的那一行数据淘汰。

（3）LRU 法

cache 控制器设置访问标志以记录数据在某一段时间内被访问的次数。当需要淘汰时，则选择近期最少使用的那一行数据淘汰。

cache 的替换算法极其重要，因为它直接影响 cache 的性能。好的替换算法能提高 cache 的命中率，从而提高系统的整体性能。

5. cache 存储器的写操作

因为需要保证缓存在 cache 中的数据与主存中的内容一致，相对读操作而言，cache 的写操作比较复杂，常用的有以下几种方法：

（1）写直达（write through）：当要写 cache 时，数据同时写回主存储器，有时也称为写通。

（2）写回（write back）：CPU 修改 cache 的某一行后，相应的数据并不立即写入主存储器单元。而是当该行被从 cache 中淘汰时，才把数据写回到主存储器中。

（3）标记法：对 cache 中的每一个数据设置一个有效位。当数据进入 cache 后，有效位置 1；而当 CPU 要对该数据进行修改时，数据只需写入主存储器并同时将该有效位清 0。当要从 cache 中读取数据时需要测试其有效位：若为 1 则直接从 cache 中取数，否则从主存中取数。

1.2.5 虚拟存储器

计算机里的程序和数据通常都存放在外存储器（辅助存储器）上，直到 CPU 需要时才调入到主存储器中。虚拟存储系统的作用是给程序员一个更大的"虚拟"的存储空间，其容量可远远超过主存储器的容量，而与辅助存储器容量相当。

虚拟存储器的管理原则与 cache 基本相似：

- 存储系统由小容量的高速存储器和大容量的低速存储器组成；
- 把要经常访问的数据驻留在高速存储器中；

• 一旦这些数据访问频率下降则把它们送回低速存储器中；

• 设计有效的管理算法，以使系统访问速率接近高速存储器，而容量接近大容量的
低速存储器，有较高的性能价格比。

1.虚拟存储器概述

在使用虚拟存储器体系的系统中，由程序（CPU）使用的访存地址称为虚拟地址，程序（CPU）直接访问的存储空间称为虚拟地址空间；而主存储器的地址则称为物理地址。通常虚拟地址空间远远大于主存储器容量。在早期没有虚拟存储器结构的计算机中；大小超过主存容量的程序往往不能直接运行，需要由程序员在设计程序时使用覆盖技术，即只把将要执行的程序片段调入内存中执行，当需要执行其他片段时，再调入所需片段覆盖通不需要的部分。这样设计大型程序或访问大规模数据时很不方便。

虚拟存储器概念的引入，使得程序员不再受限于主存储器的容量，可以使用很大的虚拟地址空间而不必再为程序的调入／移出费心，辅助存储器和主存之间的数据交换由虚拟存储系统自动实现。虚拟存储器的工作过程如图 1.14 所示。

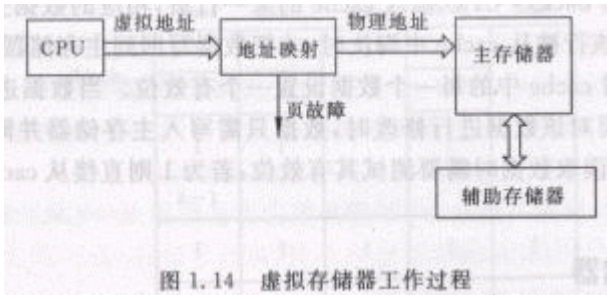


图 1.14 虚拟存储器工作过程

CPU 发出访存请求，其提供的地址为虚拟地址，需要经过地址映射以确定所需内容是否已在主存储器中。若已调入主存储器，则把虚拟地址映射为物理地址来访问主存；否则产生页故障，系统把相应的数据从辅存调入主存，同时再送往 CPU。若主存内容已满，则还需淘汰掉某些数据。

受制造工艺所限，由于页故障而产生的开销相当大，差不多是页命中开销的 1000～10000 倍。与此对应，cache 未命中的开销只是 cache 命中的开销的 4～20 倍。所以对虚拟存储器的管理策略明显不同于对 cache 的管理策略。在 cache 未命中期间，由于时间不长，CPU 只能作一些与表维护有关的操作，等待从主存取数。而页故障延迟时间一般约为 1～100ms，若闲置 CPU 则显然对 CPU 的利用率会大大下降。所以在早期的虚拟存储系统中，为了充分利用处理机周期，常常采用多个任务共享处理器的策略。当一个任务产生页故障后，处理器转去执行另一个任务，因而大大提高了处理器的利用率，但同时也增加了对每个用户的响应时间。现在的虚拟存储系统在产生页故障后通常不再把 CPU 的控制权移交给另一作业，而是让 CPU 等待从辅存调人新的页面以避免竞争处理器。

2.虚拟存储器的地址映射机制

地址映射的基本功能是实现从虚拟地址到物理地址的转换。我们以采用页面管理方式的存储器为例，介绍虚拟存储器的地址映射机制。页面管理系统把主存和虚存空间分割成固定大小的页面（如 1024 字节），分别称为物理页面和虚拟页面（逻辑页面）。

如图 1.15 所示，虚拟地址分成两部分：虚拟页号 VP 和页内地址 VD；同样物理地址也分成两部分，物理页号 PP 以及页内地址 PD。通常虚拟地址空间的页面与主存储器空间的页面大小相等，但虚拟地址空间远大于主存空间。

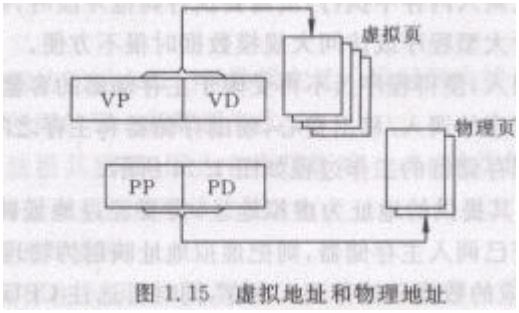


图 1.15 虚拟地址和物理地址

在页式管理系统中建立页表以存放虚拟页号和物理页号的对应关系。利用页表就可以实现地址映射功能。与 cache 存储器类似，常用的映射方式有 3 种：全相联映射、直接映射和组相联映射。

当所提供的虚拟存储器空间很大时，页表的尺寸也将随之增加，从而加大处理的难度。如虚拟地址空间为 4GB（2^{32}=4G），而页面大小为 1K 时，则页表将有 4M 项。以现在的计算机系统配置，不太可能把页表存放于主存中。通常采用二级地址映射方案代替一级映射来处理这样庞大的页表。

在二级映射机制中设置了高速缓存作地址变换后备缓冲器（translation lookaside buffer，TLB），又称"快表"，来减少开销。典型的 TLB 二级虚拟地址映射机制如图 1.16 所示。

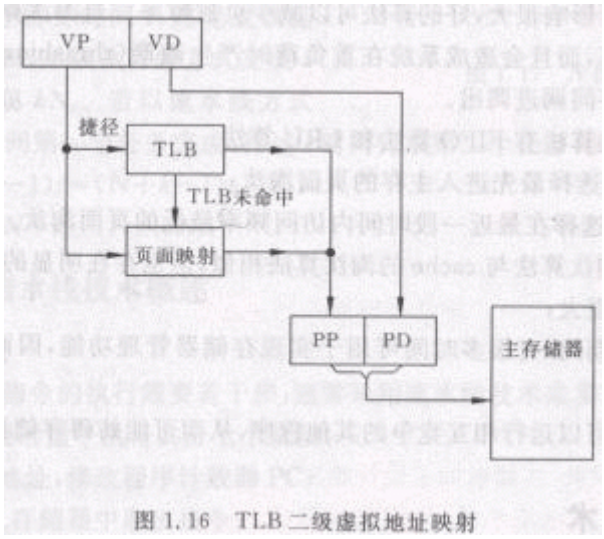


图 1.16 TLB 二级虚拟地址映射

虚页号先通过捷径送往 TLB 以检查 TLB 中是否有其映像，若有则直接从 TLB 返回其对应的物理页号 PP，PD 直接由 VD 可得：PD=VD，地址变换结束。若 TLB 未命中则从主存中访问页表，经过地址映射得到物理页号 PP，同时把该 VP 和 PP 送入 TLB。为了提高 TLB 查找速度，往往使用相联存储器技术。

3.分段存储系统

以上介绍的虚拟存储器地址映射都是基于页面管理系统的。页面是虚拟存储器系统中使用的最小单元，因此为了访问某一存储单元（字，word）就需把包含该单元的整个页面全部调入主存。通常页面的划分由虚存管理系统负责，页面与程序逻辑结构无关。这样一页中的内容往往逻辑上无关，调入该页常会引入许多无用内容，从而降低系统性能。

分段存储器以段作为基本单元，段是相关程序和数据的集合，形成一个子程序，所以分段存储器系统能够反映程序内部的逻辑关系。段与页不同，段长是不固定的，由程序员设计。而且分页地址空间是一维空间，所有地址都是连续的；分段存储器是二维地址空间，由段号+段内位移量组成其地址，段内地址连续而段与段之间不连续。可变段长的分段存储管理比固定页面大小的分页存储管理要复杂得多。

可以把分页和分段技术结合起来组建虚拟存储器系统：用分段表示程序及数据的逻辑结构，然后把段再分成固定大小的页面。

4.页面调度算法

当主存已满而又需从辅存中调入新页面时，就要从主存中淘汰一个页面。页面淘汰算法对虚存系统性能影响很大，好的算法可以减少页故障率。当淘汰算法选择不当时，不仅可能增加页故障率，而且会造成系统在重负荷时产生颠簸（thrashing）--即某些页面频繁地在主存和辅存间调进调出。

常用的页面淘汰算法有 FIFO 算法和 LRU 算法。

• FIFO 算法：选择最先进入主存的页面淘汰；

• LRU 算法；选择在最近一段时间内访问频率最低的页面淘汰。

虚拟存储器的淘汰算法与 cache 的淘汰算法相似，但也存在明显的差异：

• 页故障代价很大；

• 响应页故障期间，有很多时间可用于实现存储器管理功能，因而可能减少将来的故障；

• 页故障期间可以运行相互竞争的其他程序，从而可能妨碍存储器管理。

3 流水线技术

流水线技术是通过并行硬件来提高系统性能的常用方法，其基本思想在冯·诺依曼第一台存储程序计算机中已经提出。60 年代硬件价格相对较高，流水线技术主要用于巨型机，如 IBM 的 STRETCH 和 CDC6600。自从 80 年代以来，硬件价格迅速下跌，使得流水线技术得到了广泛应用。

1.3.1 流水线技术的基本原理

流水线技术其实是一种任务的分解技术。把一件任务分解为若干顺序执行的子任务，不同的子任务由不同的执行机构负责执行，而这些机构可以同时并行工作。在任一时刻，任一任务只占用其中一个执行机构，这样就可以实现多个任务的重叠执行，以提高工作效率。

流水线技术早已应用于社会生产活动的各个领域。在完成一个老任务之前就可以启动一个新任务，满负荷后流水线完成任务的速度只取决于提供新任务的速度而与该任务所需的全部时间无关。

典型的流水线操作如图 1.17 所示。从图中可以看出，由于多个任务可以重叠地执行，虽然完成一个任务的时间与单独地执行该任务相近（由于分段的缘故，可能多一些），但是从整体上看完成多个任务所需的时间则大大减少。



图 1.17 N段流水线

假定有某种类型的任务，共可分成 N 个子任务，每个子任务需要时间 t，则完成该任务需要时间 N_1。若以单独执行的方式完成 k 个任务，则共需时间 k N_1。

若以流水线方式执行，则经过 N_1时间第一个任务完成，再过 t 时间完成第二个任务，所以完成 k 个任务花费时间为：

N_1＋（k-1）t=(N＋k－1)t 。

当 k 较大时，（N＋k-1）t《k N_1。

1.3.2 计算机流水线技术概述

1.指令流水线

计算机中一条指令的执行需要苦干步，通常采用流水线技术来实现指令的执行，以提高 CPU 性能。典型的指令执行共分七个阶段：

（1）计算指令地址，修改程序计数器 PC；

（2）取指，即从存储器中取出指令；

（3）指令译码；

（4）计算操作数地址；

（5）取操作数；

（6）执行指令；

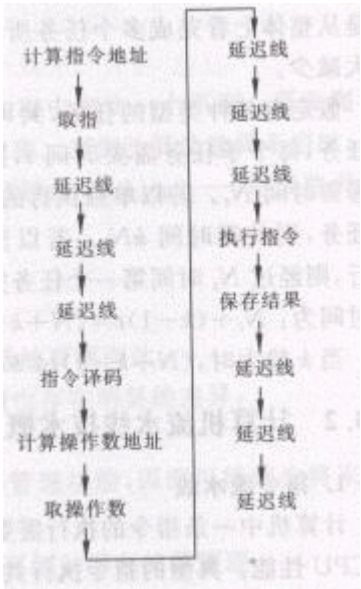
（7）保存结果。

对指令执行阶段的划分也可以把取指作为第一阶段，其他阶段顺序前移，而在最后一个阶段计算下一条指令的地址。

若假定指令执行的各个阶段的执行时间相同，都是一个周期。执行一条指令就需要花费七个周期的时间。采用流水线技术以后，当满负荷时，每个周期都能从流水线上完成一条指令，相当于性能大约改善了七倍。

实际上，流水线技术对性能的提高程度取决于其执行顺序中最慢的一步。例如，在指令执行的七个阶段中，如果访问存储器需要四个周期，而其他操作只需一个周期，一条指令的执行共需访存三次再加上四个单周期的执行段，所以共需要 16 个周期。采用流水线以后，由于受限于访存操作，四个周期才能完成一条指令的执行，因此性能提高了大约四倍。

如图 1.18 所示，通过在流水线中引入延迟线，每段延迟线延迟一个周期，进一步细化访存操作，从而可以提高流水线的效率。



1.18 增加了延迟线的指令流水线

该流水线长度增加了，然而却能每周期启动一条指令，完成一条指令。性能比使用单一部件顺序执行快 16 倍。但从该流水线也可以看出，当其满负荷运行时，任何时刻都有 12 个不同的段对存储器进行访问，这就要求存储系统能支持 12 个独立的并发访问而不能相互冲突。

流水线设计的基本思想其实很简单，任何一个可以分解的任务都可以用流水线来做，可以设置多个处理机构，分别执行相应的子任务。为了提高流水线性能，有些处理时间长的步还需分解成更小的步，使流水线上所有步的处理时间相同。

但实际上由于流水线上不同段的处理过程相互之间有影响，因此流水线的设计需要考虑多种因素来优化流水线结构，并以某种方式处理各种影响流水线性能的因素，从而增加了设计难度。

【例 1.1】设有四级流水线，分别完成取指、译码、运算、存数四步操作，各步时间依次为 100ns，50ns，80ns 和 100ns。流水线的操作周期应为多少？

解：由流水线技术基本特征可知，其平均时间取决于流水线最慢的步，所以该流水线的操作周期应为 100ns。

2.运算操作流水线

以上讨论了指令级流水线技术的概况，解释了流水线技术是怎样提高计算机性能的。 其实，计算机在执行各种运算操作时也可以应用流水线技术来提高运算速度。

例如执行浮点加法运算，可以把它分成三个阶段：对阶、尾数相加和结果规格化，其结构如图 1.19 所示。

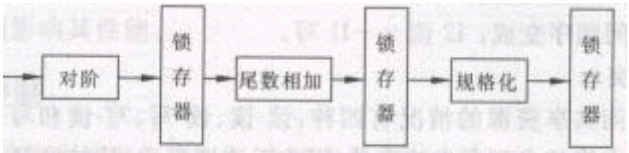


图 1.19 运算操作流水线

流水线的三个阶段用锁存器进行分割，锁存器用来在相邻两段之间保持处理的中间结果，以供下一阶段使用。这样在满负荷时，该流水线可以同时处理三条浮点加法指令。

1.3.3 影响流水线性能的主要因素

从前面的介绍中可以看出，流水线的关键之处在于"重叠执行"。为了得到高的性能表现，流水线应该满负荷工作，即各个阶段都要同时并行地工作。但是在实际情况中，流水线各个阶段可能会相互影响，阻塞流水线，使其性能下降。阻塞主要由以下两种情形引起；执行转移指令和共享资源冲突。

1.转移指令的影响

通常在顺序执行指令的情况下，当 CPU 取一条指令时，流水线的地址计算部件可以独立地把当前 PC 值加上当前指令长度来计算下一条指令的地址，从而可以并行地工作，但是当流水线执行一条转移指令时，就会引起流水线的阻塞。因为在该转移指令完成之前，流水线都不能确定出下一条指令的地址。

所以为了保证指令的正确执行，必须把取指段和指令地址计算段互锁。在取出转移指令后，立即锁住指令地址计算段，直到转移指令执行完成。互锁阶段流水线处于等待状态，不能满负荷工作，因而性能下降。

2.共享资源访问冲突

当多条指令以流水线方式重叠执行时，由于可能会引起对共享的寄存器 / 存储器资源访问次序的变化，因此将导致冲突--这种情况又称为数据相关。

（1）访问次序的变化

为了简单起见，设有顺序执行的两条指令 I1 和 I2。I1 的运算结果存放在存储单元 M1 中，而 I2 则需要从 M1 中取出 I1 的运算结果作为源操作数。其正常执行顺序为 M1：I1 写--I2 读。然而，如果以流水线方式执行 I1 和 I2，则如图 1.20 所示。

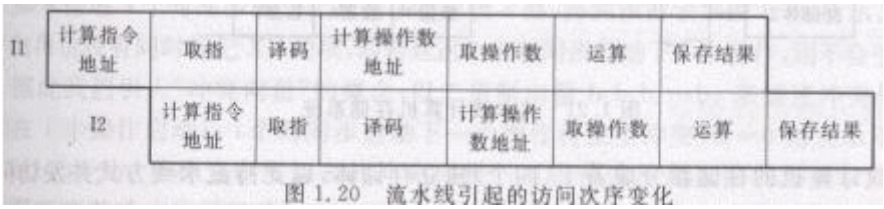


图 1.20 流水线引起的访问次序变化

如果不对流水线做某些控制，可以看出当 I2 取操作数时 I1 还没有保存结果，所以实际上对 M1 的访问顺序变成：I2 读--I1 写。

（2）访问冲突

两个操作访问共享资源的情况有四种：读-读、读-写、写-读和写-写。除了读-读组合不会因为使用流水线技术而产生冲突外（因为无论谁先读，其结果都一样），其余三种都会产生冲突：

①读-写：由于流水线引起执行次序的变化，先写而后读则要读的数据已被破坏；

②写-读：如果因为流水线而变成了先读后写，则读的数据是无效的旧数据；

③写-写：如果两个写操作的顺序发生了变化，则实际写入存储单元的数据也就会不同，因而产生错误。

为了避免冲突，就需要把相互有关的指令进行阻塞，这样就会引起流水线效率的下降。一般说来指令流水线级数越多，越容易导致数据相关，阻塞流水线。

1.3.4 流水线计算机的存储器结构

流水线技术使处理器能够并行地重叠执行若干条指令，从而提高了指令的执行速度。但这又要求其存储系统能够及时地响应流水线的访存需求，而且要求存储系统能够支持并发访问而不产生冲突。指令流水线技术其实是把处理瓶颈从 CPU 子系统转移给了存储子系统。

在存储系统中也需要使用流水线技术以匹配采用了流水线技术的 CPU 系统的处理能力。常见的流水线计算机的存储系统如图 1.21 所示。

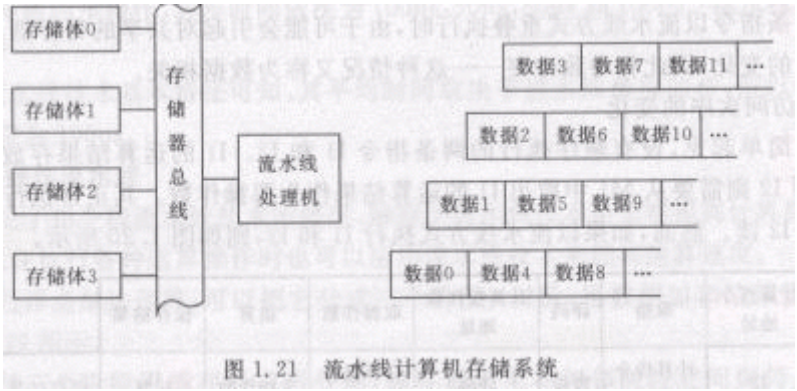


图 1.21 流水线计算机存储系统

流水线计算机的存储器分成若干（四个）独立存储体，以支持流水线方式并发访问。

流水线计算机中也使用了 cache，通常分为指令 cache 和数据 cache，各自用于存放指令和操作数。在通常情况下该结构工作良好，但是如果系统允许程序在执行过程中修改其自身，就会大大地影响其性能。

1.3.5 流水线控制

为了尽可能地使流水线满负荷工作，在流水线入口处设置一个控制器，用以控制流水线入口处的各种操作。控制器在入口处不断接收新操作进入流水线，使流水线一直维持在最大的吞吐率，同时又要确保在流水线各段不会产生冲突。

1971 年 Davidson 提出用预留表来预测冲突。预留表是从流水线设计直接推导出来的一张表，表中列出了流水线上各部件操作的时间信息，每一行代表流水线中的一段，而每一列则代表一个时间步。如图 1.22 所示为一张预留表：有标记的项表明在该时间步相应的部件已被占用（预留）。

	1	2	3	4	5	6	7
A	X						
B		X	X				
C			X	X			
D					X		X
E						X	

图 1.22 预留表

	1	2	3	4	5	6	7
A	X	Y					
B		X	XY	Y			
C			X	XY	Y		
D					X	Y	X
E						X	Y

图 1.23 延迟 1 个时间步启动下一个操作

从预留表可以看出，该流水线一共有 5 段：A～E，完成一个操作共需 7 个时间步。

把预留表做成一个模板，把它覆盖在另一相同的预留表上，用移动模板表示启动一个操作。例如，我们把模板向有移动一个时间步表示在一个操作启动后隔一个时间步再启动下一个操作。以 X 标记第一个操作，Y 标记第二个操作，得到结果如图 1.23 所示。如把模板向右移动两格，即表示延迟 2 个时间步启动下一个操

作，其预留表如图 1.24 所示。从这两张预留表中可以发现有些项同时被标记，说明在该时间步两个操作都需要同一个部件，即发生了冲突。这就表示在启动 1 个操作后，延迟 1 或者 2 个时间步启动下一个操作都会引起冲突。

如果推迟 3 个时间步呢？让我们移动模板 3 格，得到预留表如图 1.25 所示。从表中可以看出再也没有同时标记 XY 的项，即若延迟 3 个时间步启动下一个操作，则不会引起冲突。

	1	2	3	4	5	6	7
A	X		Y				
B		X	X	Y	Y		
C			X	X	Y	Y	
D					X		XY
E						X	

图 1.24 延迟 2 个时间步启动下一个操作

	1	2	3	4	5	6	7
A	X			Y			
B		X	X		Y	Y	
C			X	X		Y	Y
D					X		X
E						X	

图 1.25 延迟 3 个时间步启动下一个操作

据此我们引入"冲突向量"的概念，以二进制向量 $b_1b_2b_3\cdots b_M$ 来描述冲突信息，b_i= 1 表示在 1 个操作启动后 i 个时间步启动下一个操作将发生冲突， b_i= 0 则表示不会发生冲突。 $b_1b_2b_3\cdots b_M$ 被称为该预留表的冲突向量。可以很方便用上面的方法求出图中所给定的预留表的冲突向量为 110000。

1.3.6 流水线的中断处理

流水线计算机处理中断的方法通常有两种：

（1）不精确断点法：当发生中断后，计算机并不立即响应该中断；而是先禁止指令再进入流水线，然后等已在流水线中的所有指令执行完毕，才响应该中断。

（2）精确断点法：当有中断请求时，立即中止当前流水线的作业，去响应中断。这种方法硬件电路比前一种方法复杂得多，但对中断响应的实时性却比前一种方法好。

4 输入输出系统

外部设置经过输入输出（I/0）系统与计算机相连。I/0 系统向外设发布控制命令决定外设要执行的操作，例如向 I/0 系统发达数据或接收从 I/0 系统来的数据以及他一些设备特定的操作。

输入输出系统是计算机与外界进行数据交换的通道，但它不仅仅只是把外部设备和系统总线简单连接起来，而且还具有某些"智能"，即负责处理外设和总线通信控制。随着计算机技术的发展，外设的种类日益增加，其控制方式也各不相同，很难做到由 CPU 来负责和这些设备直接打交道。外设的数据传输率千差万别，把高速的系统总线和低速的处设直接相连也是很不实际的。同时外设的数据格式也不一定会和计算机相同，因此无法直接进行数据交换。所以需要一个 I/O 系统在 CPU 主存储器和各种外部设备之间负责协调和控制数据的输入输出。

可以毫不夸张地说，没有 I/O 系统的计算机几乎没有任何用处.

1.4.1 输入输了控制器

1. 输出输现控制器基本功能概述

输入输出控制器负责控制外部设备与主存储器、CPU 的寄存器之间的数据交换。I/O 模块用以控制一个或多个外设与 CPU、主存储器进行数据交换。它通过系统接口（内部接口）与主机（CPU、主存储器）交互，通过设备接口（外部接口）与各种外设打交道。

（1） 控制与定时（control & timing）：I/O 系统必须对通信进行控制和定时，以协调外设和内部资源之间的通信流量，因为内部资源如主存、系统总线等被一系列的活动包括数据的 I/O 所共享。

一个典型的外设向 CPU 传送数据的例子如下：

① CPU 向 I/O 模块查询外设之状态；

② I/O 模块返回该外设之状态；

③ 如果外设已准备好发送数据，则 CPU 发出控制命令给 I/O 模块要求开始传递；

④ I/O 模块从外设取得数据

⑤ I/O 模块把该数据发给 CPU。

同时 I/O 模块和 CPU 之间的传递还需进行总线仲裁。以解决多个设备争用总线的问题。

（2）与 CPU 通信（CPU communication）

①命令：I/O 系统必须能够接收和翻译由 CPU 发来的控制命令；

②数据：I/O 系统还需通过数据线与 CPU 交换数据；

③状态；I/O 系统能够向 CPU 根据该状态信息来完成数据交换；

④ 地址：每一个外设都具有自己的地址，I/O 系统应该能够识别它所控制下的每一个外设的地址。

（3） 与设备通信（device communication）：I/O 系统与外设之间交换控制命令、状态信息以及数据。

（4） 数据通信缓冲（data buffering）：I/O 系统需要数据缓冲寄存器来暂存数据，以此来适应通信双方速率的不匹配，从而提高主机利用率。

（5） 错误检测机构（error detection）：外设和主机进行数据交换的过程中可能会产生错误，通常使用奇偶校验码、CRC 循环冗余校验码进行检测和纠正。在主机和外设之间设置 I/O 系统，其目的在于隐藏各种外设的上体细节信息（如数据格式、时序等），给 CPU 提供了一个统一、简便的控制界面用以完成数据交换，以减轻 CPU 的 I/O 负担。

2. 输入输出系统的基本结构如图 1.26 所示。

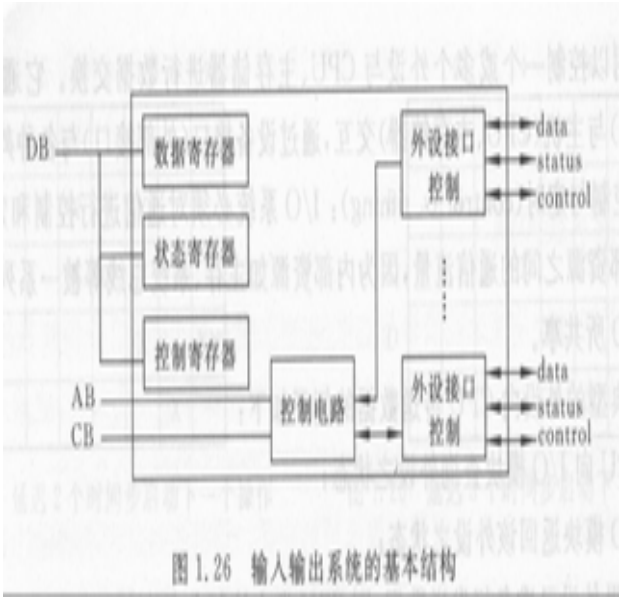


图 1.26 输入输出系统的基本结构

（1） 数据寄存器：用于缓冲从 CPU 来的或是从外设来的数据。

（2） 状态寄存器：记录设备当前状态，CPU 通过查询该寄存器可知道设备情况，以决定操作顺序。

（3） 控制寄存器：CPU 通过对该控制器进行访问来决定该外设执行的操作和控制（控制外调以及 I/O 系统）。

（4） 控制电路：I/O 系统的核心。负责控制 I/O 系统的运行，同时还要完成外部设备地址的识别以确定 I/O 操作将发生在哪个外设上。同时可能还需要中断控制逻辑，直接存储访问（direct memory access DMA），控制逻辑以支持中断方式或 DMA 方式通信。

（5） 外设接口控制：接受 CPU 发来的控制命令，控制外设与 I/O 之间的数据交换等。

1.4.2 输入输出的工作方式

输入输出系统主要有三种方式与主机交换数据：

（1） 程序控制方式（programmed I/O）：输入输出完全由 CPU 控制，在整个 I/O 过程中 CPU 必须等待而不能进行其他工作，无法充分发挥 CPU 高速的处理能力。

（2） 程序中断方式（interrupt-driven I/O）：I/O 模块以中断的方式通知 CPU 开始传递数据，无需 CPU 主动查询和等待外设，从而把 CPU 解放出来做其他工作，以提高 CPU 的利用效率。

（3） DMA 方式：在该方式下，CPU 只是在数据传输前和完成后才介入，而数据的传输过程由 DMA 控制器来管理，无需 CPU 参与。数据直接写入或读出主存储器，不再经过 CPU 中转。

1. 程序控制输入输出

由程序主动查询外设，完成主机与外设间的数据传送，方法简单，硬件开销小。CPU 发出 I/O 命令，命令中包含了外设的地址信息和所要执行的操作，相应的 I/O 系统执行该命令并设置状态寄存器；CPU 不停地（定期地）查询 I/O 系统以确定该操作是否已经完成。典型的 I/O 命令可以分为三类：

- control 命令：激活某一外设，通知它应做之事；
- Test 命令：用以测试检查 I/O 系统和外设的当前状态；
- R/W 命令：用以输入和输出数据。

程序的输入输出如图 1。27 所示。

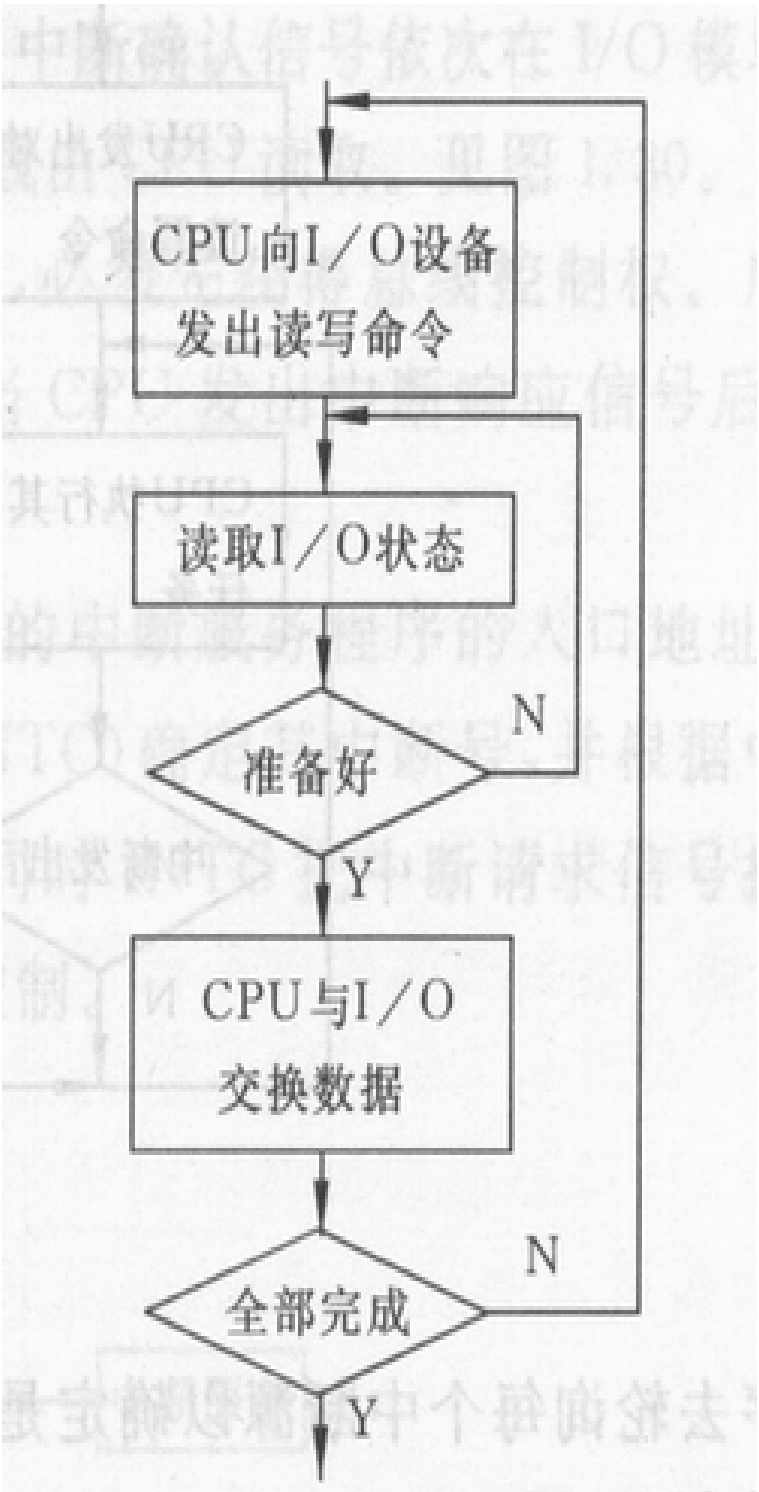


图 1.27 程序控制的输入输出

（1） I/0 设备的编址方式

① 存储器映射：即 I/O 设备和内存储器统一编址，使用相同的机器指令来访问内存和外设。

② 独立编址：I/O 设备和内存储器的地址空间相互独立，CPU 使用专门的 I/O 指令来访问外设。

（2） 多个外设的查询方式：在程序控制的输入输出方式中，当有多少个外设需要和主机交换数据时，程序（CPU）必须定期轮询外设，以确定每个外设的状态。

其查询方式也可以分为两类：

①串行点名：CPU 依次对所有的外设进行查询，每次只能查询一台。

②并行查询：把各个外设状态位集中起来，由 CPU 通过一个专用的端口来读取，每一次可以同时查询多个外设的状态。

2. 中断方式输入输出

由此程序控制 I/O 的方法其主要缺点在于 CPU 必须等待 I/O 系统完成数据传输任务，在此期间 CPU 需定期地查询 I/O 系统的状态，以确认传输是否完成。因此整个系统的性能严重下降。

为了克服该缺陷，把中断机制引入到 I/O 传输过程中。CPU 利用中断方式完成数据的输入/输出：当 I/O 系统与外设交换数据时，CPU 无需等待也不必去查询 I/O 的状态。而可以抽身出来处理其他任务。当 I/O 系统完成了数据传输后则以中断信号通知 CPU。CPU 然后保存正在执行程序的现场，转入 I/O 中断服务程序完成与 I/O 系统的数据交换。然后返回原主程序继续执行。与程序控制方式相比，中断方式因为 CPU 无需等待而提高了效率。中断方式输入输出如图 1.28 所示。

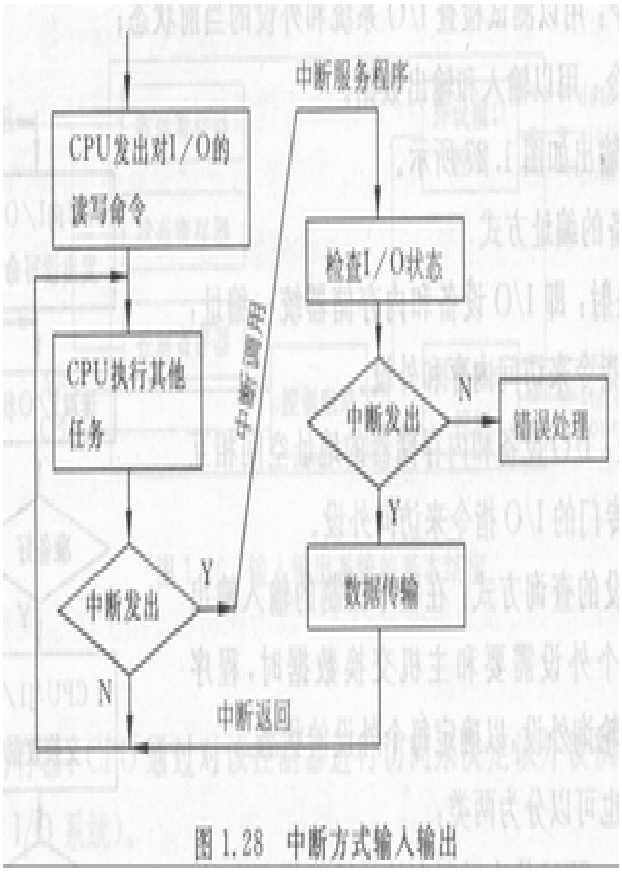


图 1.28 中断方式输入输出

在系统中具有多个中断源的情况下，常用的处理方法有：多中断信号线法（multiple interrupt lines ）、中断软件查询法（software poll）、雏菊链法（daisy chain）、总线仲裁法和中断向量表法。

（1） 多中断信号线法：每个中断源都有属于自己的一根中断请求信号线向 CPU 提出中断请求。

（2） 中断软件查询法：当 CPU 检测到一个中断请求信号以后，即转入到中断服务程序去轮询每个中断源以确定是谁发出了中断请求信号，对各个设备的响应优先级由软件设定，见图 1.29。

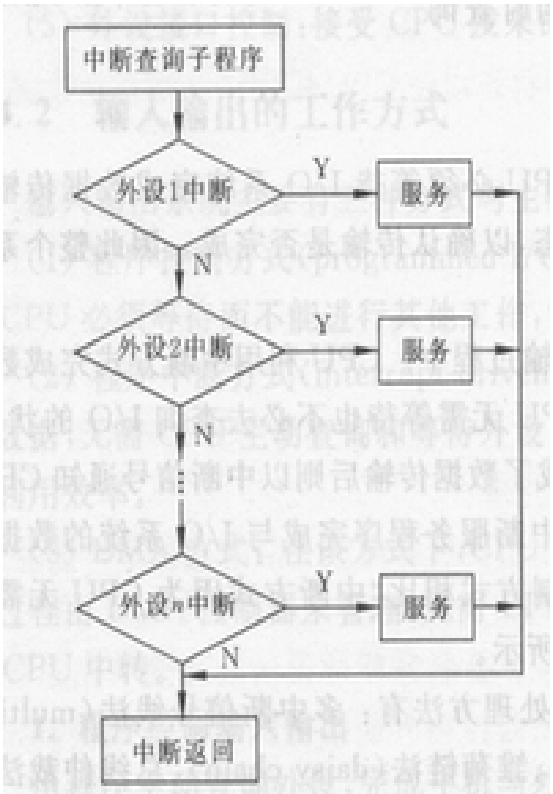


图 1.29 中断软件查询法

（3） 雏菊链法：软件查询的缺陷在地花费时间太多。雏菊链法实际是一种硬件查询法。所有的 I/O 模块共享一根共同的中断请求线，而中断确认信号则以链式在各模块间相连。当 CPU 检测到中断请求信号，则发出中断确认信号。中断确认信号依次在 I/O 模块间传递，直到发出请求的模块，该模块则把它的 ID 送往数据线由 CPU 读取。见图 1.30

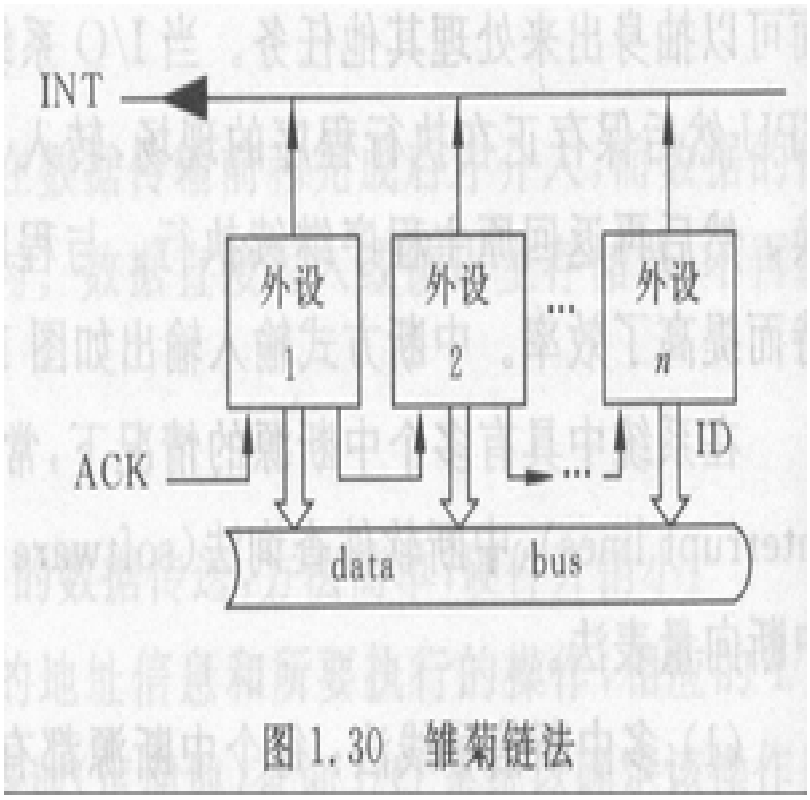


图 1.30 雏菊链法

（4） 总线仲裁法：一个 I/O 设备在发出中断请求之前，必须先获得总线控制权。所以可由总线仲裁定谁可以发出中断请示信号。当 CPU 发出中断响应信号后，该设备即把自己 ID 发往数据线。

（5） 中断向量表法：中断向量表用来保存各个中断源的中断服务程序的入口地址。当外设发出中断请求信号（INTR）以后，由中断控制器（INTR）确定其中断号。并根据中断号查找中断向量表来取得中断服务程序的入口地址，同时 INTC 把中断请求信号提交给 CPU，如图 1.31 所示。中断源的优先级由 INTC 来控制。

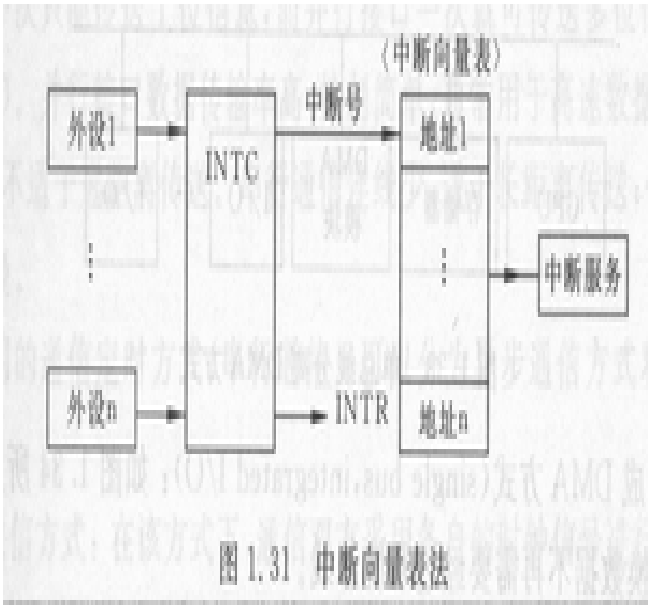


图 1.31 中断向量表法

3. DMA 方式输入输出

中断法虽然比程序控制法更加有效（也能快速响应外设的要求），但两者都是由软件来完成数据的传输，难以胜任高速的数传需求。并且在主存和 I/O 模块之间进行数据交换仍需由 CPU 控制。并且每一数据都需要经过 CPU 中转。因此不仅影响了 CPU 的工作效率也降低了 I/O 数据传输率。

DMA 方法使用 DMA 控制器（DMAC）来控制和管理数据传输。DMAC 和 CPU 共享系统总线，并且具有独立访问存储器的能力。在进行 DMA 时，CPU 放弃对系统总线的控制而由 DMAC 控制总线；由 DMAC 提供存储器地址及必须的读写控制信号，实现外设与存储器之间进行数据交换。DMAC 的基本结构如图 1.32 所示。

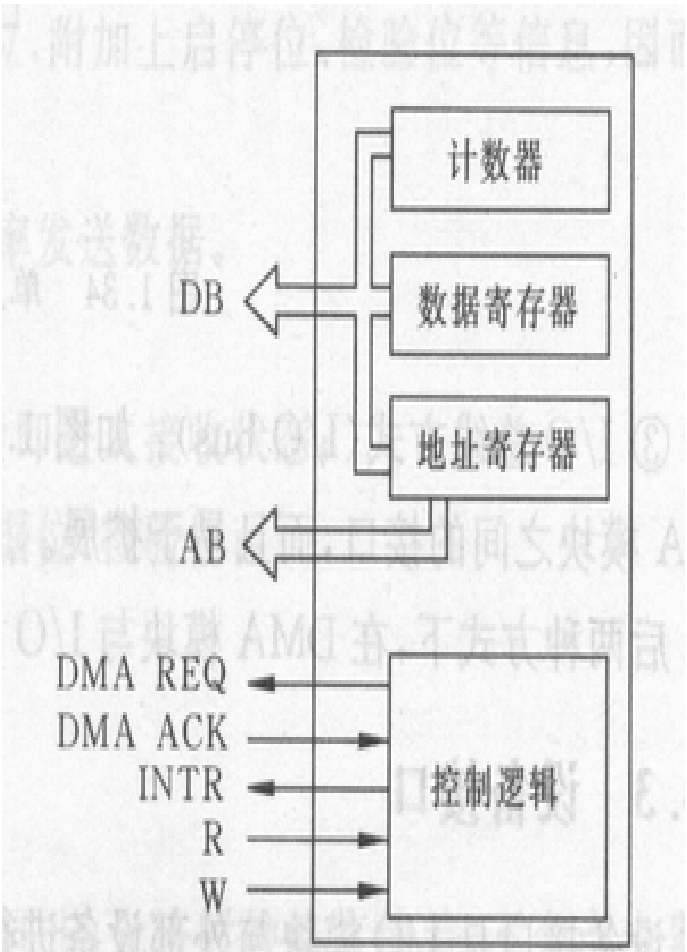


图 1.32 DMAC 的基本结构

（1） DMAC 获取总线的三种方式

①暂停方式：DMAC 请求 CPU 让出系统总线，由 DMAC 使用，直到一组数据全部传送完毕后，才把总线控制权还给 CPU。

②，周期窃取方式：请求 CPU 进入空闲态暂时放弃总线，插入一个 DMA 周期，传送完一个字后，把总线还给 CPU。相当于 CPU 暂停了一个总线周期。

③ 共享方式：在 CPU 不使用总线的时候由 DMAC 进行 DMA 传输。

（2） DMA 的三种组织方式

①单总线分离 DMA 方式（single bus ，detached DMA ）：如图 1.33 所示，由 DMA 模块代替 CPU 完成数据传输的控制任务，存储器和 I/O 模块之间的数据交换必须经过 DMA 模块。这种方案效率不高，每传一个字都需占有 2 个总线周期。

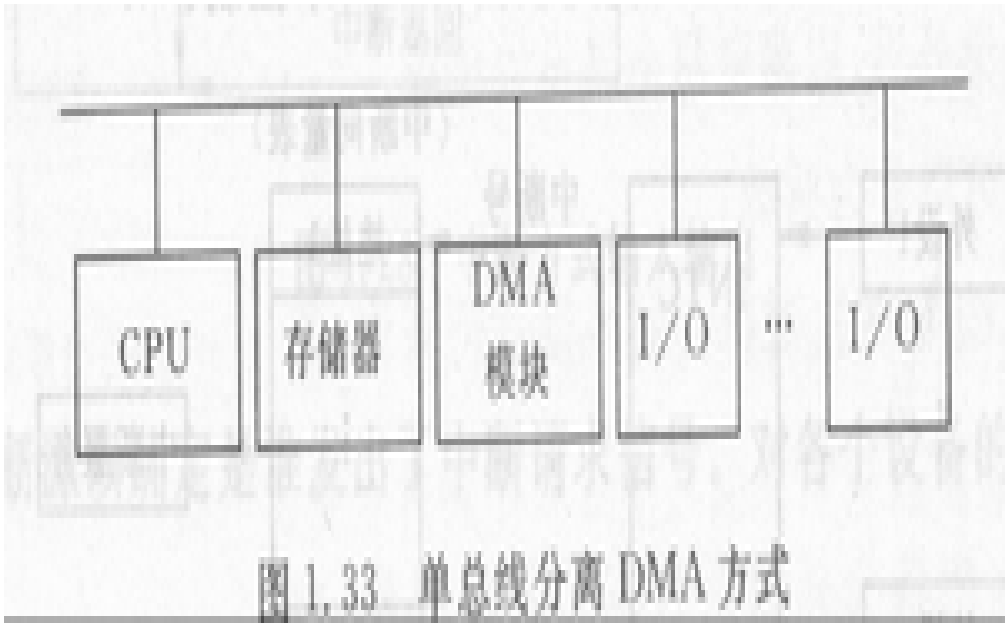


图 1.33 单总线分离 DMA 方式

②单总线集成 DMA 方式（single bus，integrated I/O）：如图 1.34 所示，在 DMA 模块和 I/O 之间交换数据不再需要经过系统总线。

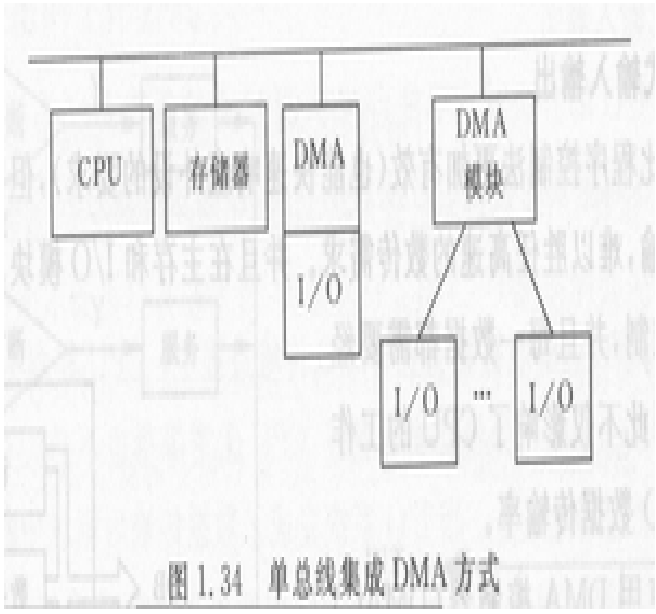


图 1.34 单总线集成 DMA 方式

③I/O 总线方式（I/O bus）：如图 1.35 所示，这种 I/O bus 方式减少了 I/O 模块和 DMA 模块之间的接口，而且易于扩展。后两种方式下，在 DMA 模块和 I/O 模块之间交换数据就不再需要使用系统总线了。

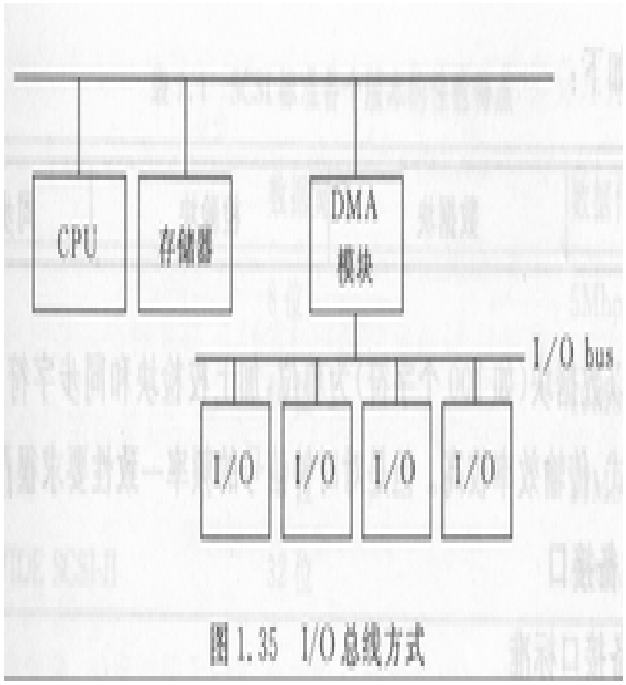


图 1.35 I/O 总线方式

1.4.3 设备接口

设备接口是 I/O 模块与外部设备进行数据交换的桥梁，用以适应各种千差万别的外部设备。

1. 接口的类型

根据外部设备与 I/O 模块交换数据的方式可以分为两种：串行联并行接口。

串行接口一次只能传送 1 位信息，而并行接口一次就可传送多位信息（如 8 位，10 位，16 位，32 位）。并行接口数据传输率高，控制简单，通常用于高速数据通道接口；但是所需连线很多，不适于远距离传送。串行通信连线少，适于长距离传送；但是控制复杂机时且传输速度较慢。

按照所采用的通信定时方式，串行通信又可分为异步通信方式和同步通信方式两种。

（1） 异步通信方式：在该方式下，通信双方采用各自的时钟信号进行定时，收发端采用相同的波特率（位/秒，bps），并约定相同的数据帧格式，为所要传送的数据添上同步控制信息，以使双方能够正确通信。其数据格式如下：

空闲位	启动位	用户数据	较验位	停止位	空闲位	启动位	.....

一般异步串行通信的数据帧是以字符为单位，附加上启停位、检验位等信息，因而附加数据较多，传输效率不高。通信过程如下：

发送方：按照约定的帧格式，以给定的波特率发送数据。

接收方：

- 检测空闲位和启动位，当发现启动位之后即进入接收状态；
- 以位时间（1/波特率）为间隔定时接收数据位、检验位；
- 进行奇偶检验；
- 进行帧定界，寻和规定的停止位；
- 如无差错，则接收该字符。

特点如下：按字符传送；在一短时间内要求同步，允许时钟略有偏差，不会产生累积效应；由启停位提供同步信息。

（2） 同步通信方式：通信双方采和同一时钟信号进行定时。

其数据格式如下：

同步字符	数据块	检验块	同步字符.....

其数据帧是以数据块（如 100 个字符）为单位，加上校检块和同步字符。因而附加数据远少于异步方式，传输效率较高。但是对时钟信号的频率一致性要求很严格。

2. 常见的设备接口

（1） 磁盘设备接口标准

① ST506 接口：该接口主要用于使用同步电机作定位机构的温盘，结构简单，只完成磁盘信息的读写放大，把数据的编码解码、数据的格式转换等功能都留给由 I/O 模块处理。其传输速率为 5Mbps-7Mbps，最多可支持 2 个硬盘，最大支持盘空间为 150MB。

② ESDI（enhanced small device interface）接口：一种通用的标准接口，不仅适用于小型温盘，还适用于磁带机和光盘存储器。该接口除了完成信息的读写放大外，还要完成数据的编码解码。数据传输率 5Mbps-10Mbp，最多可支持 4 个硬盘，硬盘空间最大可达 600MB

③ IDE（integrated drive electronics）接口：该接口是微机中常用的接口标准，分为普通 IDE 和增强型 IDE（EIDE）接口。

• 普通 IDE：数据传输率不超过 1.5Mbps，数据传输宽度为 8 位；最多可连接 4 个 IDE 设备；每个 IDE 硬盘容量不超过 5280MB。

• EIDE 接口:数据传输率可达 12Mbps~18Mbps,数据传输宽度 32 位;最多可连接 4 个 IDE 设备;每个 IDE 硬盘可超过 528MB.

④ SCSI(small computer system interface)接口:自从 1984 年 SCSI 接口应用于 Macintosh 机后,现在它已经作为一种系统级的设备接口,广泛广泛应用于 Macintosh 机,以及许多工作站.SCSI 是并行接口,其数据宽度为 8 位、16 位和 32 位，它是大容量存储设备、音频设备和 CD-ROM 驱动器的一种标准。SCSI 接口通常被看作是一种总线，可用于连接多个外设，这些 SCSI 设备以雏菊链形式接入，并被分配给一个惟一的 ID 号（0-6），其中 7 号分配给 SCSI 控制器。某些 SCSI 控制器可以提供多达 35 个 SCSI 通道。SCSI 设备彼此独立运作，相互之间可以交换数据，也可以和主机进行交互。数据以分组消息（message packet）的形式进行传输。与上述几个接口相比，SCSI 接口承担了更多的数据传输控制任务。

SCSI 标准各个版本的性能特点如表 1.1 所示。

表 1.1 SCSI 标准各个版本的性能特点

版本号	数据宽度	数据传输率
SCSI-I	8 位	5Mbps
SCSI-II	16 位	10Mbps
Fast SCSI-II	16 位	20Mbps
FAST/WIDE SCSI-II	32 位	40Mbps

（2） 打印机设备级接口：打印的接口有并行和串行两种，目前打印机的接口多采用并行方式，其标准如图 1.36 所示。



图 1.36 并行打印机接口

（3） PCMCIA（personal computer miniature communications interface adapter）接口：PCMCIA 是一种广泛用于笔记本电脑的接口标准，体积小，扩展较方便灵活。最初 PCMCIA 主要用于笔记本电脑扩展内存，目前常用作一种存储器卡接口或进行 fax/modem 功能扩展接口。现在用 PCMEIA 代表个人计算机储器卡国际协会（personal computer memory card international association），而 PCMCIA 接口更名为 Pccard 接口。

（4） P1394 串行接口：随着处理器速度的激增以及主机和外设之间需要交换的数据量的膨胀，I/O 系统的任务越来越艰巨。一方面要改进 I/O 模块的结构以适应高速海量数据交换的需求，另一方面也需设计一种性能优异的设备接口代替 SCSI 等接口以减轻 I/O 模块的压力。P1394 接口（ANSI 标准）就是这样一种高速的串行总线，用以连接众多的外部设备。

P1394 有许多优于 SCSI 等其他外设接口的特点：数据传输率高、价格低且容易实现，所以不仅应用于计算机系统中，也广泛用于消费类电子产品，诸如数字像机、VCD 等。P1394 接口连接形式如图 1.37 所示。

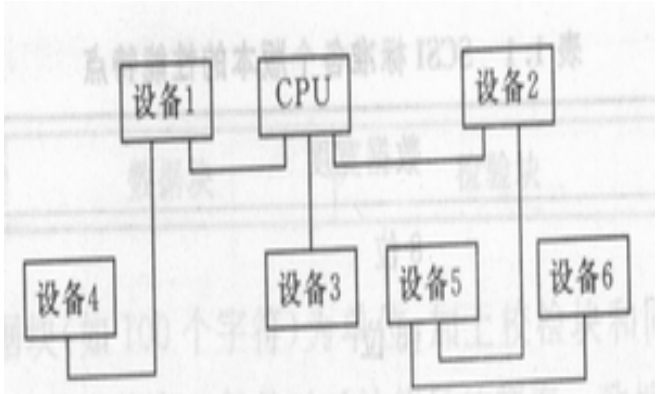


图 1.37 P1394 接口的连接形式

P1394 接口的一个优越之处在于串通信方式。因为并行通信并行接口（如 SCSI）需要更多的信号线，所以成本及故障率都高于串行通信接口。而且信号线之间也需屏蔽相互之间的干扰，并且随通信距离的增加，其信号间的同步也会变得比较困难，P1394 因为使用了串行通信方式，尺寸较小，所以适合当前计算机功能日益强大而体积日益变小的趋势。

P1394 提供一个单一的接口用以连接众多的外部设备例如鼠标、激光打印机、外部磁盘存储器等。它使用了一种雏菊链式的设备连接方式，一个端口可以支持 63 个设备；而且使用桥（bridge）互联的方式，以树型结构配置，可以支持的设备数高达 1022。P1394 支持设备的热插拔。更换设备时无需断掉计算机电源。P1394 支持自动配置，无需手工设置设备的 ID，也无需关心设备间的相对位置。

P1394 的一个重要的思想是引入了分层的体系结构，使主机与外设通过串行总线实现了互连的标准化。它一共具有三个层次：

• 物理层（physical layer）：定义 P1394 标准支持的传输介质以及电气特性等；

• 连接层（link leyer）：描述数据以分组形式进行传输的过程；

• 处理层（transaction layer ）：定义一种请求-应答协议，以隐藏 P1394 低层实现的细节。

1.4.4 输入输出系统的发展

伴随着计算机系统的日新月异，其 I/O 系统也逐渐从简单到复杂，担任了越来越多的 I/O 控制任务，把 CPU 从繁杂的 I/O 控制任务中解放出来，以提高系统的整体性能。I/O 系统的演化大致经历了以下六个阶段：

（1） 数据通路：CPU 直接控制外设，I/O 系统仅仅表现为数据的形式。

（2） 程序控制：I/O 系统增加了对数据交换的控制功能，CPU 使用程序控制方法（programmed I/O）来进行数据的读写。由于 I/O 系统担任了部分控制功能而使 CPU 从某种程度上不再需要关心外设的具体细节。

（3） 中断方式：把中断机制引入 I/O 数据交换过程，从而大大减少了 CPU 的等待时间，提高了效率。

（4） DMA 方式：使用 DMA 方法进行数据传输，进一步解放了 CPU。在传送一块数据时，除了开始和结束的时刻都不再需要 CPU 的参与，而由 DMAC 来完成控制任务。

（5） I/O 通道：I/O 系统的功能进一步发展，使用专门的处理器来完成控制功能。该 I/O 处理器具有专为外设定制的指令集，以适应各种 I/O 所需。数据的 I/O 控制由 CPU 启动 I/O 处理器执行，I/O 处理器执行相应的 I/O 程序完成数据交换而无需 CPU 介入。当 I/O 处理器完成了任务之后，再以中断方式通知 CPU。

（6） I/O 处理机：I/O 系统不仅拥有专门的 I/O 处理器，而且配置了本地存储器--在某种程度上已经是一台计算机了。强有力的配置使得 I/O 系统可以控制大量的外部设备，执行大部分控制功能，进一步减少了 CPU 介入与外设的数据交换。

5 RISC 计算机

处理器指令集的设计是计算机体系结构设计的一个极其重要的环节。在早期的计算机里，由于硬件设计的限制，处理器的指令系统很小，指令也很简单，以减少硬件电路，降低成本。随着电子技术的发展，特别是 VLSI 技术的出现，硬件成本不断下跌，计算机的指令系统亦随即膨胀起来。现在大部分的计算机都拥有 100 条以上甚至超过 200 条的指令。

为了提高操作系统的效率，缩短指令系统与高级语言之间的语义差别，以便于高级语言的编译，降低软件成本，人们最初选择了向指令系统中添加更多、更复杂的指令的方法。这些指令支持更多的数据类型和寻址方式。同时为了提高应用的性能表现，人们往往把原本由软件实现的功能改由硬件实现；而为了保持和以前同一系列的机器的程序兼容，指令系统中的指令只增不减。这些都促使计算机的指令系统越来越庞大、复杂。这种类型的计算机通常被称为复杂指令集计算机（complex instruction set computer ,CISC）。例如 DEC 公司的 VAX11/780 计算机拥有 303 条指令，支持 18 种寻址方式。

与庞大复杂的指令系统相伴而来的是日益复杂的硬件实现，研制周期、成本以及风险都随之增加。并且由于各种指令种类繁多，它们之间的差异很大，难以实现流水线操作，反而不利于改善提高系统性能。

从 70 年代中期开始，人们开始日益关注指令系统的合理性问题。IBM 公司的 Johncocke 提出了精简指令系统的想法。研究人员发现，计算机指令系统如果使用少量结构简单的指令会提高计算机性能，因为这些指令可以在 CPU 内更快地执行。这种类型的计算机因为其指令数量少、结构简单，而被称为精简指令集计算机（reduced instructionset computer，RISC）。

1.5.1 RISC 计算机的基本原理

RISC 计算机不是简单地把指令系统进行简化，而是通过简化指令的途径使计算机的结构更加简单合理，以减少指令的执行周期数，从而提高运算速度。所以有人把它称为 RICC（reduced instruction count computer）。

计算机执行程序所需的时间 P 由三方面因素决定：高级语言程序编译后产生的机器指令数 I，执行每条指令所需的平均周期数 CPI 以及每个机器周期的时间 T。

P=I×CPI×T

由于 RISC 指令系统比较简单，所以编译后产生的 RISC 指令数要大于相应的 CISC 指令数；但是 RISC 结构简单，其每个机器周期的时间小于 CICS 的每个机器周期的时间；并且随着 RISC 硬件结构的改进，大部分分指令可以在一个（或者小于一个）机器周期内完成，其 CPI 远小于 CISI 的 CPI；因此 RISC 机上程序所需的执行时间小于该程序在 CISC 机上的运行时间。

1.5.2 CISC 计算机与 RISC 计算机的主要特点

1. CISC 计算机的主要特点

（1） 指令数量众多：CISC 计算机，"机如其名"，其指令系统拥有大量的指令，通常有 100-250 条左右。在设计指令系统时，不仅需要考虑到机器语言才能执行。设计复杂指令系统的一个主要目的就是为了简化编译过程，以提高系统整体性能。

（2） 指令使用频率相差悬殊：CISC 计算机指令使用频率相差悬殊，最常使用的是一些比较简单的指令，仅占指令总数的 20%，但在程序中出现的频率却占 80%。而大部分复杂指令却很少使用。

（3） 支持很多种寻址方式：CISC 机指令系统支持的寻址方式通常为 5~20 种。

（4） 变长的指令格式:CISC 机指令系统一个重要的特点就是其指令长度不是固定的。例如使用寄存器操作数的指令其长度可能为 2 个字节，而带 2 个存储器地址的指令可能占用 5 个字节。变长的指令增加了指令译码电路的复杂性。

（5） 指令可以对存储器单元中数据直接进行处理：典型的 CISC 处理器通常都有指令能够直接对内存单元中的数据进行处理。例如一条加法指令可以完成把两面三刀下内存单元中的数据进行加法运算，然后把和保存到内存单元中去的功能。这样一第指令的执行就需要进行 3 次存储器访问操作，其执行速度较慢。

2. RISC 计算机的主要特点

（1） 指令数量少：RISC 机优先选取使用频率最高的一些简单指令以及一些常用指令，避免使用复杂指令。典型的 RISC 机指令系统中的指令大多数都是执行寄存器-寄存器操作；对存储器的操作仅提供了读和写两种方式；所有的计算都有是在 CPU 寄存器里进行。这种体系结构化了指令系统，并且促进了寄存器操作的优化。

（2） 指令的寻址方式少：RISC 机通常支持寄存器寻址方式、立即数寻址方式以及相对寻址方式。

（3） 指令长度固定，指令格式种类少：因为 RISC 指令数量少，格式也相对简单（与 CISC 比较），其指令长度固定，指令之间各字段的划分比较一致所以其译码也相对容易。

（4） 只提供了 Load/Store 指令访问存储器：与 CISC 支持直接对存储器单元里的数据操作不同，RISC 指令集中只提供了从存储器读数 Load 和把数据写入存储器 Store 两条指令来与存储器打交道。其余所有的操作都在 CPU 的寄存器间进行。

（5） 以硬布线逻辑控制为主：RISC 机为了提高操作的执行速度，通常采用硬功夫布线逻辑（组合逻辑）来构建控制器。而 CISC 机的指令系统很复杂，难以用组合逻辑电路实现控制器,通常采用微程序控制。

（6） 单周期指令执行：RISC 机因为简化了指令系统，所以很容易利用流水线技术使得大部分指令都能在一个机器周期内完成。少数指令可能会需要多个周期执行，例如 Load/Store 指令因为需要访问存储器，其执行时间就会长一些。

（7） 拥有相当多的寄存器：RISC 机拥有相当多的寄存器，可以用于存放中间数据和优化对操作数的访问，从而提高了处理器的性能。例如寄存器中某一数据需要写回存储器保存，而稍后又需要从存储器中再把该数据读出。如果寄存器数量众多，则可以把该数据保留在寄存器里，下次就直接从寄存器中读取而无需访存，从而加快程序的执行速度。

（8） 优化的编译器：RISC 的精简指令集使编译工作简单化。因为指令长度固定、格式少、寻址方式少，编译时不必在具有相似功能的许多指令中进行选择，也不必为寻址方式的选择而费心，同时易于实现优化，从而可以生成高效率执行的机器代码。

1.5.3 RISC 处理器结构概述

采用 RISC 技术的 CPU 硬件一般具有如下特点：寄存器数量多；采用流水线组织；控制器的实现采用硬布线控制逻辑电路。

1. RISC 通用寄存器

增加寄存器的数目可以减少程序访问存储器的次数。因为在程序运行时，相当部分的指令用于访问存储器（据统计，在 CISC 中 50%以上指令是用于访存的）。研究表明，采用大量的寄存器对 RISC 体系性能的改善程序不亚于使用精简指令集对性能的改善。

RISC 机寄存的组织有两种方案：

（1） 重叠寄存器窗口技术（overlapped register windows ,ORW）；

（2） 设计一套寄存器分配算法，并通过编译程序的优化处理来充分利用寄存器资源。

下面简单介绍重叠寄存器窗口技术。

所谓寄存器窗口实际上就是一组寄存器。当程序执行过程调用与返回时，RISC 处理器通过应用重叠寄存器窗口技术，利用寄存器组而不是存储器来完成参数传递，并且不再需要保存和恢复寄存器内容，因此大大提高了程序的执行速度。根据统计，在 C 语言程序中，Call/Return 语句占语句总数的 12%，而每对 Call/Return 又会访存多次（保存/恢复寄存器内容），所以应用重叠寄存器窗口技术可以显著提高系统性能。

ORW 技术中，其寄存器被分为两类：全局寄存器和窗口寄存器。如图 1.38 所示，系统工程共有 64 个寄存器，其中 R_0～ R_7用作全局寄存器 R_8～ R_{63}，用作窗口寄存器。

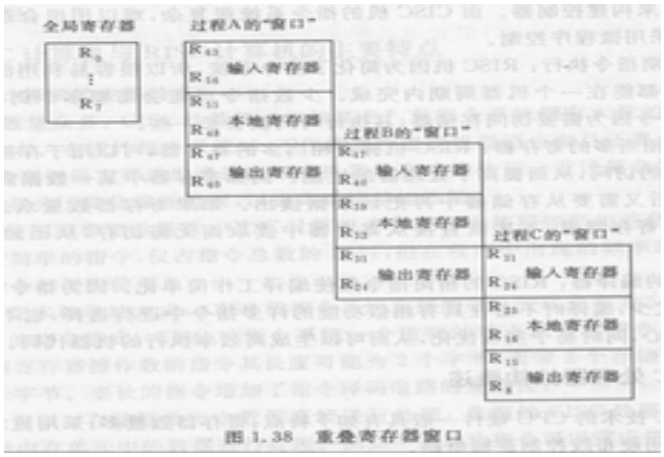


图 1.38 重叠寄存器窗口

图 1.38 中共有 3 个过程 A，B，C，其调用次序为 A 调用 B 而 B 又调用 C。 R_0～R_7 为 A，B，C 共享。 R_8～ R_{63}共分成 3 个窗口（即寄存器）分配给 3 个过程使用：A 的窗口为 R_{40}～ R_{63}，B 的窗口为 R_{24}～ R_{47}，C 的窗口为 R_8～ R_{31}。

每个过程分配了一个具有 24 个寄存器的窗口，其中 24 个寄存器又分成 3 部分：输入寄存器组、输出寄存器和本地寄存器组。例如过程 A 的输入寄存器组为 R_{56}～ R_{63}，本地寄存器组为 R_{48}～ R_{55}，输出寄存器组为 R_{40}～ R_{47}。本地寄存器组用于保存属于该过程的局部数据，输出寄存器组与被过程调用的子过程输入寄存器组相重叠，而输入寄存器组与调用该过程的父过程的输出寄存器相重叠。

当前程序所访问的窗口由一个指针 CWP 标示。当调用新过程时，修改 CWP 以指向新窗口，该过程返回时，又再次修改 CWP 指向原来的窗口。

当过程 A 调用过程 B 时，由于两者使用不同的物理寄存器，所以不必保存现场，而过程 B 的输入寄存器组与 A 的输出寄存器组重叠（ R_{40} ～ R_{47}），所以 B 可以直接取得参数。当从 B 返回 A 时，B 把返回参数送入 B 的输入寄存器组，A 则可以直接从自己的输出寄存器组取得返回参数。

程序的嵌套层次受窗口数限制，当超过规定时会产生溢出。所以这时就需要把其中一窗口内容送入存储器保存起来，当用到该窗口时，再从存储器中读出。

2. RISC 流水线

（1） 取出：RISC 通过使用流水线技术来实现每个机器周期执行一条指令。由于存储器速度低于 CPU 的速度，为了使 RISC 的指令流水线在每个机器周期都能取得一条指令，通常使用指令预取技术或者 cache 来加快读取指令的速度。

① 指令预取技术：在 CPU 中设置指令寄存器堆，指令预先从存储器取到寄存器堆中。只要存储器空闲和寄存器堆未满，就不断从存储器中取出指令。所以当执行取指令操作时，就能很快从寄存器堆中取出。但当程序执行程序顺序改变时，会使预取的指令失效，从而增加了存储器的读写频率，以致影响总线效率（尤其是当程序执行顺序频繁改变时）；并且因为等待从储存器中取出的新指令，将会暂时阻塞流水线。

② cache 方案：大多数 RISC 采用了 cache 方案，使用 cache 来提高取指的速度。而且有的 RISC 甚至使用两个独立的 cache 来改善性能；一个称为指令 cache，另一个称为数据 cache。这样取指和读数可以同时进行，互不干扰。

（2） 数据相关的解决方法：解决流水线中数据相关的一种方案是增加 bypass 路径。例如第 n 条指令和第 n+1 条指令均为加法指令，而第 n 条指令的输出结果要用作第 n+1 条指令的输入数据。当第 n+1 条指令取数时，第 n 条指令的结果还没有送回目的寄存器，这样就产生了数据相关。在 ALU 的输出端和输入端寄存间建立 bypass 路径，将第 n 条指令的结果直接从输出端送往输入端寄存器作为第 n+1 条指令的源操作数，从而可以避免系统性能的下降。

（3） 条件转移指令的处理：由于硬件方法无法解决由于执行条件转移指令而推迟执行下一条指令的取出所引起的性能损失，RISC 采用编译优化，在转移指令后插入一条任何跳转都必定执行的指令，以保持流水线畅通。

3. 硬布线逻辑控制

由于 RISC 简化了指令系统，指令数量少且格式较简单，各指令所需的机器周期数相差不大，每个机器完成的操作也较简单，因而没有必要采用微程序控制。

与 CISC 相比，RISC 中产生控制信号的电路仅占很小面积，多余下来的面积可用于增加更多的寄存器或其他的功能电路。

1.5.4 RISC 的编译优化

RISC 计算机的程序执行效率与编译程序密切相关的，编译程序需要做到：合理安排寄存器的使用，以减少访存次数；调整指令的执行顺序，避免或减少数据相关；提供高效的例行程序；常规的编译优化。

1. 转移指令的处理

通常 RISC 采用延迟转移的方法。编译程序通过调整程序指令来解决因为转移指令而引起的流水线阻塞。假定有 3 条指令Ⅰ1，Ⅰ2 和Ⅰ3，Ⅰ3 为转移指令。其在 CISC 机器上执行顺序为Ⅰ1→Ⅰ2→Ⅰ3。若在 RISC 也按同样顺序生成指令流，则执行到Ⅰ3 时然会引起流水线阻塞。所以编译程序调整其顺序为
Ⅰ1→Ⅰ2→Ⅰ3。在 RISC 执行时处理器知道转移指令后面的一条指令是肯定要执行的，所以在执行跳转指令Ⅰ3 时，CPU 无需阻塞，可以直接去取指令Ⅰ2。

2. 减少流水线的互锁

减少流水线互锁的常用方法为：

（1） 两条相邻指令若产生数据相关，则设法在这两条指令间插入其他的指令隔开它们；

（2） 把访存指令隔开，尽量插入不进行存储器读写的指令，以避免存储器发生访问冲突而等待；

（3） 在相邻的浮点指令间插入别的指令，以避免由于浮点指令执行时间长而产生阻塞（若浮点运算部件亦采用流水线技术，则不需要）。

1.5.5 典型的 RISC 处理器

自从80年代初美国加州大学伯克利分校的RISCⅠ、RISCⅡ以及斯坦福大学的MPS机问世以来，RISC 技术得到了迅速的发展。当今最具影响力的 5 种 RISC 芯片如下：

1. DEC 公司的 Alpha 21164

（1） 采用超标量技术。

（2） 片内含有 2 个整数运算部件和 2 个浮点运算部件，允许同时并行工作。

（3） 每个机器周期可执行 4 条指令。

（4） 整数运算采用 7 级流水线，浮点运算采用 9 级流水线。

（5） Alpha 21164 具有 3 级 cache 结构：第一级包括 8KB 的指令 cache 和 8KB 的数据 cache，第二级 cache 容量有 96KB，片内还设有第三级 cache 的控制器，其 SRAM 在片外。访存时 CPU 首先访问第一级 cache，若未命中则由硬件自动访问第二级 cache，若还未命中则访问第三级 cache。只有当第三级 cache 也未命中时才访问主存。

（6） 支持指令的转移预测：当遇上转移指令时，处理器对其转移方向进行猜测，根据猜测按某一分支往下执行程序，但把结果暂存在缓冲器内。当猜测正确时再把结果送到指令中指定的目的地去，当猜测错误时，就需要为之付出代价：为了恢复到程序的转移点再取出要执行的指令通常需要 4 个时钟周期。

2. IBM 公司的 Power PC620

（1） 采用超标量技术。

（2） 片内有 6 个独立的执行单元：3 个整数运算部件；1 个浮点部件；1 个转移部件，用以执行转移操作；一个装入/存储部件，用以完成读取和保存数据。

（3） 二级 cache：第一级 cache 位于芯片内，分为 32KB 指令 cache 和 32KB 数据 cache；第二级 cache 控制器位于芯片内，外接 SRAM，其数据总线宽达 128 位。

（4） 支持指令的转移预测。

（5） 超过 5 级的流水线。

（6） 在软件控制下既可使用 Motorola 的编址方式，也可以使用 Intel 编址方式。Power PC620 可以运行以 Motorola 方式编址的操作系统，再在该操作系统上运行按 Intel 方式编址的应用程序。

3. HP 公司的 PA-8000

（1） 超标量结构。

（2） 功能部件包括 ALU、移位/合并部件、乘加部件、2 个浮点运算部件、2 个装入/存储部件。

（3） 支持指令的转移预测。

（4） 可以并行执行 4 条指令。

4. SGI 公司的 MIPS 分部的 TS

（1） 超标量结构。

（2） 功能部件包括 2 个整数运算部件、2 个浮点运算部件和 1 个装入/存储部件。

（3） （超过）5 级的流水线。

（4） 可以并行执行 5 条指令。

（5） 支持指令的转移预测。

（6） 二级 cache 结构：第一级 cache 位于片内，包括 32KB 指令 cache 和 32KB 数据 cache;片内还设有第二级 cache 控制器,可外接 512KB-16MB 的二级 cache。

5. SUN 公司的 Ultra SPARC

（1） 超标量结构。

（2） 功能部件包括 2 个整数运算部件、2 个浮点/图形加运算部件、2 个浮点/图形除运算部件、浮点除运算部件、转移部件和装入/存储部件。

（3） 9 级流水线。

（4） 每周期可以执行 4 条指令。

（5） 支持指令转移预测。

（6） 二级 cache 结构：第一级包括 16KB 指令 cache 和 16KB 数据 cache；片内还设置了第二级 cache 控制器。

（7）支持紧耦合多处理和松耦合多处理两种体系结构,在紧耦合多处理系统中，不用增加逻辑电路就可以互联 2-4 个处理器;在松耦合多处理器系统中可以互联欢会 128 个处理器。

目前 RISC 处理器技术的发展方向是采用并行处理技术幅度提高运算速度。超级流水线（superpipeline）、超级标量（superscalar）以及超长指令（VLIW）等并行处理技术使得 RISC 微处理器的性能进一步得到了提高。

6 并行处理技术

1.6.1 并行处理技术概述

1. 并行性的概念

指令流水线之所以能够提高指令的执行速度（如 RISC 机上每个机器周期都可以在流水线上完成一条指令的执行），其关键在于指令的重叠执行。与非流水线方式相比，每条指令的执行时间并没有变化，但执行多条指令所需的总时间却大大减少了。

实现指令的重叠执行，实际上就是把一条指令的执行分成顺序的若干段，而流水线处理器能够同时执行若干条指令的不同阶段--流水线结构通过引入并行处理技术提高了系统的性能。

所谓并行性（parallelism）就是指在同一时刻或者同一时间间隔内完成两种或两种以上性质相同或不同的工作，只要时间上互相重叠，就都蕴含了并行性。例如：假定各元器件延迟相同，使用 n 位运算器进行 n 位并行运算的速度近似为使用 1 位运算器进行 n 位串行运算速度的 n 倍，这就是并行性的体现；再如操作系统中采用的多道程序分时运行也是并行性的体现。

严格地说，并行性实际上具有同时性和并发性两重含义：

- 同时性（simultaneity）：两个或两个以上事件在同一时刻发生；

- 并发性（concurrency）：两个或两个以上事件在同一时间间隔内发生。

并行处理反映了计算机系统结构向高性能发展的趋势：一方面在单处理机内广泛采用多种并行性措施；另一方面则摆脱了单处理机的束缚，发展各种不同形式的多机系统，以达到更高的并行处理水平。

2. 并行性的层次

并行处理（parallel processing）是信息处理的一种有效形式，它着重发掘解题过程中的并行事件，使并行性的层次提高，以使系统性能更上一层楼。并行性有不同的层次，而且从不同的角度看，其层次结构也不一样。

（1） 程序执行的并行性：从程序执行的角度来看，从低到高并行性可以分为四个层次。

①指令内部并行：一条指令内各个微操作之间的并行。

②指令间并行：多条指令的并行执行，需要解决指令间存在的相互关联。

③ 任务或进程间并行：关键在于如何进行任务分解。

④ 作业或程序间并行：关键在于设计并行算法把有限的软、硬件资源有效地同时分配给正在用于求解一个大题目的多个程序。

并行性层次从低到高反映了硬件实现的比例在下降。例如在过去的单处理机中并行性上升到任务层或作业层时，主要是通过操作系统中的进程管理、作业管理以及并行语言、并发程序设计等软件方法实现。随着硬件成本的迅速下降和软件成本的上升，并行性的实现逐渐增大了硬件比例，如现在的多处理机系统中其并行性主要由硬件实现。

（2） 数据处理的并行性层次从低到高依次为：

①位串字串：一次只对一个字的一位进行处理（无并行性）。

②位并字串：一次对一个字的全部位进行处理。

③ 位串字并：一次对许多字的同一位（位片）进行处理。

④ 位并字并（全并行）：对许多字的全部或部分进行处理。

（3） 操作并行性层次如下：

①存储器操作并行：可以在一个存储器周期内访问多个存储单元。

②处理操作步骤并行：指令的取码、译码、取操作数、执行等操作步骤在时间上重叠执行；也包括各种运算的操作步骤的重叠执行。

③ 处理器操作并行：设置大量的处理单元，在同一控制器控制下按同一条指令的要求对多个数据组同时操作。

3. 并行性措施

可以采用多种途径来提高计算机系统的并行性。常用的措施可以分为三类：时间重叠、资源重复和资源共享。

（1） 时间重叠（time interleaving）：在并行性概念中导入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部件，以加快硬件周转，提高处理速度。这种方法原则上不需要重复增加硬件设备就可以提高计算机系统的性价比。最简单的例子就是前面讨论过的指令流水线。

时间重叠措施是单处理机提高系统并行性的重要方法之一，其基本原则是功能专用化（functional speeialization）。即把一种工作按功能分割成若干相互联系的部分，每一部分由专门的部件来执行，然后按流水线原则把各个部分的执行过程在时间上重叠起来，使所有部件依次分工合作。单处理机不断地对功能部件进行分离、细化，并且注意克服信息流通中影响速度的瓶颈来提高并行性：

• I/O 瓶颈：为了缓和慢速的 I/O 设备对高速 CPU 速度的不匹配而造成的系统性能下降，采用 I/O 通道甚至 I/O 处理机来完成 I/O 设备与 CPU 之间隔数据交换。

• 存储器瓶颈：由于 CPU 的速度高于主存储器的速度，在从主存中读数或向主存写入数据时 CPU 需要等待而产生延迟。为了解决这一瓶颈，采取了多种措施：增加通用寄存器，设置 cache，使用并行主存系统等等。

• I/O 存储器瓶颈：为了匹配主存和 I/O 系统的速度，而在 I/O 系统中增加智能控制器或者是专门的数据库机。

采用流水线技术可以进一步提高单处理机的并行性。分别在指令步骤和操作步骤上实现流水线结构。流水线还可以进一步发展到处理机一级，形成以任务重叠为特征的宏流水线，这就是由处理机发展到多处理机系统--非对称或异构多处理机系统。例如，系统的测试诊断、终端信息的预处理、数据库管理、数组运算等，都分别由专用处理机进行处理，而各处理机之间以流水线方式处理以提高整个系统的性能。

（2） 资源重复（resource replication）在并行性概念中导入空间因素，基本于"以数量取胜"的原则，通过重复设置硬件资源来提高系统可靠性或性能。例如双工系统就是利用资源重复，使用两台相同的计算机完成同样的任务来提高可靠性。但随着硬件价格的下降和芯片中设立多个功能部件等。

单处理机也通过采用资源重复的措施来提高并行性。先是由按位串行处理升级到按字并行，继之则设置多操作部件以及多体存储器来提高性能，在此基础上又更进一步发展到相联处理机、并行处理机等 SIMD 结构的多机系统--同构或对称多

处理机系统。对称式系统具有模块化的结构，便于使用 LSI、VLSI 电路实现，系统扩展方便。

（3） 资源共享（resource sharing）：多个用户按一定时间顺序轮流地使用同一套资源，以提高其利用率，从而相应地提高整个系统的性能。例如分时系统就是利用共享 CPU、主存资源等来降低系统价格，提高设备利用率。

采用资源共享方式提高并行性，使得单处理机向分布式处理系统方向发展。在单处理机上首先采用多道程序和分时操作以实现资源共享。单处理机多道程序分时操作从其原理上看，实际就是用单处理机模拟多处理机功能。基于这种思路又陆续发展了虚拟存储器、虚拟处理机等。随着远程终端、计算机网和微型机的发展，虚拟的处理机被真正的处理机所代替，构成以分散为特征的多处理机系统代替以集中为特征的分时系统。这种大量分散、重复的处理机资源互联起来，在操作系统全盘控制下统一而和谐地工作而很少依赖于集中的程序、数据或硬件的系统被称为分布式处理系统。

1.6.2 并行处理机

1. 并行处理机的基本原理

并行处理机有时也称为阵列处理机（array processor），其得名于在单一控制部件控制下的由多个处理单元构成的阵列。并行处理机使用按地址访问的随机存储器（RAM），以单指令流多数据流（SIMD）方式工作。主要用于要求大量高速进行向量或矩阵运算的应用领域。

并行处理机制并行性来源于资源重复，把大量相同的处理单元（processing unit，PE）通过互联网络（interconnection network，ICN）连接起来，在统一的控制器（CU）控制下，对各自分配来的数据并行地完成同一条指令所规定的操作。PE 是不带指令控制部件的算术逻辑运算单元。

并行处理机是依靠操作级的并行处理为提高系统性能的。它以 SIMD 方式工作，其指令的执行特点是二级重叠：

（1） 并行处理机中的指令可以分为两类，只适合串行处理的控制和标量类指令由控制器本身执行，而把适合于并行处理的指令"播送"到所有处理单元。让那些处于"活跃"的状态的处理单元并行执行，而不活跃的处理单元则不参与执行。这就实现了串行类指令（标量、控制指令）与并行类指令（向量指令）的重叠执行。

（2） 并行处理机中的指令部件采用流水线方式工作，可以进一步实现多条向量指令间的重叠执行。

2.并行处理机的结构

根据存储器的不同组织形式，并行处理机有两种典型的结构。

（1）具有分布式存储器的并行处理机：具有分布式存储器的并行处理机的基本结构如图 1.39 所示。

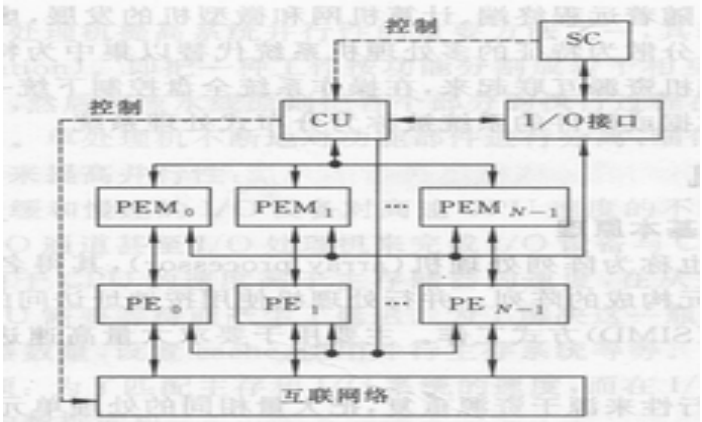


图 1.39 具有分布式存储器的并行处理机结构

①局部存储器（processing element memory ，PEM）：每个处理单元 PE 都设置有局部存储器 PEM，用以存放分布式数据，它只能被本处理单元直接访问。

②管理处理机（SC）：通常是一种通用计算机，用于管理整个系统的全部资源，完成系统维护、输入输出、用户程序的汇编及向量化编译、作业调度、存储分配、设备管理、文件管理等操作系统的功能。

③ 互联网络（ICN）：在 CU 控制下，把各 PE 互联起来，以进行信息交换。

整个系统是在控制器统一控制下运行系统程序和用户程序的。用户程序执行时，所有的指令都在控制器中进行译码，若是串行类指令则就由 CU 直接执行；若是向量类指令则被播送给所有 PE 并行执行，通过使用屏蔽位向量来控制各个 PE 的状态（活跃/不活跃）。

为了有效地对向量数据进行高速处理，关键是数据应在各个 PEM 之间合理分配，使各个 PE 主要依靠自身存储器 PEM 中的数据进行运算，尽量减少 PE 之间的数据交换。PE 之间的数据交换可以通过 ICN 进行，也可以先把数据从 PEM 读至 CU 再播送给全体 PE。

60 年代美国伊利诺尹大学和 Burroughs 公司联合研制的 ILLIAC-IV 并行处理机、1979 年美国 Goodyear 宇航公司研制的巨型并行处理机 MPP(massively parallel processor）都采用了这种结构。

（2） 具有集中共享存储器的并行处理机：具有集中共享存储器的并行处理机的基本结构如图 1.40 所示。集中共享存储器并行处理机与分布式存储器并行处理相比，其结构上的一个主要差异在于前者的存储系统帅一个共享的我体并行存储

器M_0 ， M_1…， M_{K-1}组成的。该存储器经过互联网络 ICN 为全部 N 个处理单元 PE_0， PE_1，…， PE_{N-1}所共享。

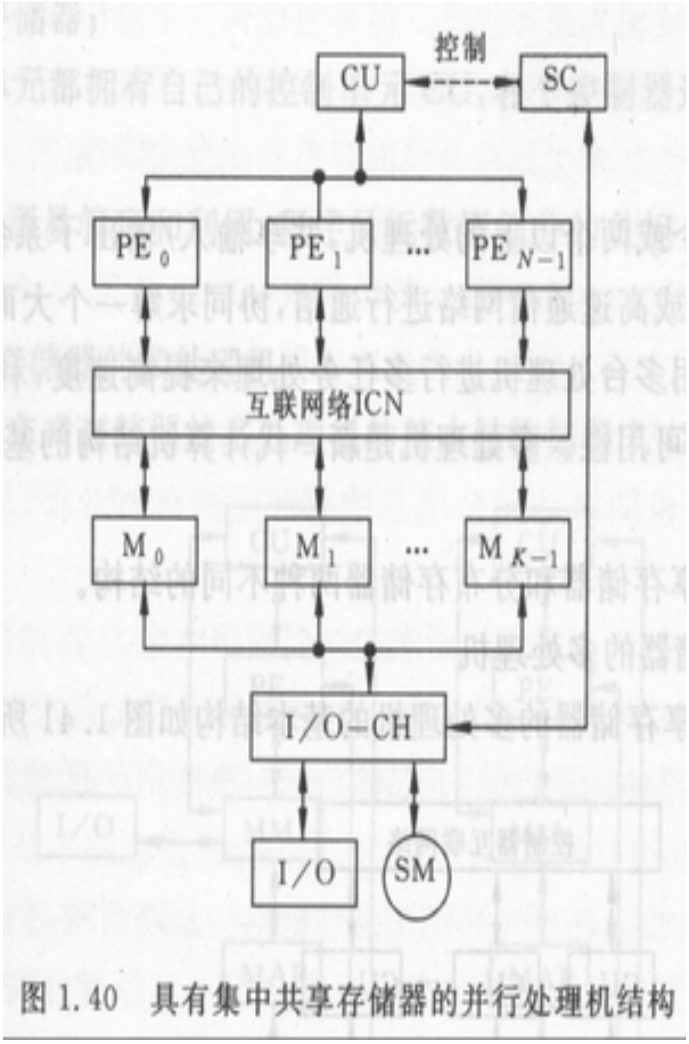


图 1.40 具有集中共享存储器的并行处理机结构

在这种结构中，为了使各个处理单元对长度为 N 的向量中的各个元素都能同时并行处理，一般情况下要求 K>N。为了尽可能避免在和处理单元访问主存时，发生存储器分体冲突，需要精心设计算法把数据按一定规律优化地分配到各个存储体中去。

集中式与分布式存储器并处理机结构上的另一个差异是 ICN 的作用有所不同。在集中共享存储器并行处理机中，ICN 是用来连接各个处理单元和存储体之间的数据通路，让各个处理单元能够以高速、灵活的方式访问不同的存储体。其结构相对而言要复杂一些，在有些并行处理机上把它称为对准网络（alignment network）。

3. 并行处理机的特点

（1） 强大的向量运算能力：基于 SIMD 方式的并行处理机是以诸如有限差分、矩阵、信号处理、线性规划等一系列计算问题为背景发展起来的。这些计算问题的共同特点是可以通过各种途径把它们转化成为对数组或向量的处理，而并行处理机利用多个处理单元对向量或数组所包含的各个分量同时进行运算，从而易于获得极高的处理速度。

（2） 并行方式：并行处理机的并行性来源于资源重复而不是时间重叠，利用的是并行性中的同时性而不是并发性。并行机提高运算速度的主要方法是增加处理单元的个数，因而潜力也较大。

（3） 适用于专门领域，如矩阵运算、微量运算等。并行处理机通常被看作是一种专用计算机，它是以一定数量的专门算法为基础的。并行处理机处理单元的阵列结构又是为某种特定类型计算问题而设计的专用结构。

（4） 标量运算速度对系统性能和影响也很重要。

（5） 具有向量化功能的高级语言编译程序有助于提高并行处理机的通用性，以及减少编译时间。

1.6.3 多处理机

多处理机具有两个或两个以上的处理机，共享输入/输出子系统，在统一的操作系统控制下，通过共享主存或高速通信网络进行通信，协同求解一个大而复杂的问题。

多处理机通过利用多台处理机进行多任务处理来提高速度，利用系统的重组能力来提高可靠性、适应性和可用性。多处理机是新一代计算机结构的基本特征。

1. 多处理机结构

多处理机具有共享存储器和分布存储器两种不同的结构。

（1） 具有共享存储器的多处理机

① 结构：具有共享存储器的多处理机的基本结构如图 1.41 所示。

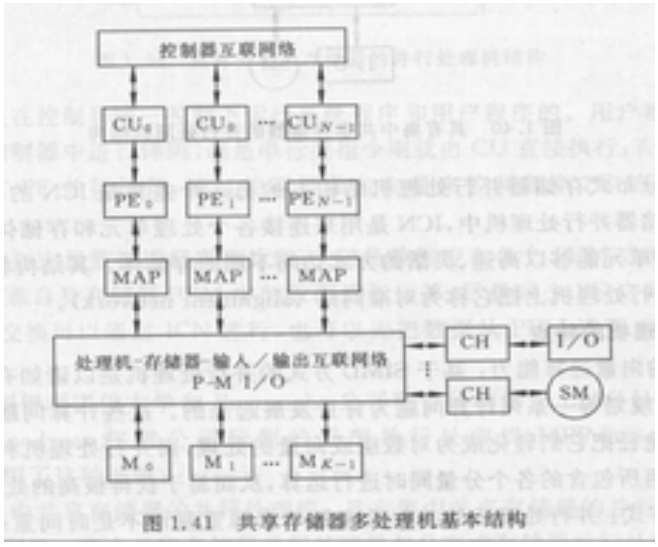


图 1.41 共享存储器多处理机基本结构

• 多个处理单元（PE_0 ～ PE_{N-1}）通过互联网络 P-M-I/O 共享集中的主存储器；

• 主存储器由多个并行的存储体（M_0 ～ M_{K-1}）组成，通常 K>N；

• 存储映射部件（MAP）：通过 MAP 把全局逻辑地址映射为局部物理地址，控制互联网络 P-M-I/O 寻找合适的路径，使得每一个 PE 都可以灵活地访问任一个存储体 M；

• 多路通道（CH）：I/O 和外存储器 SM 通过 CH 与互联网络 P-M-I/O 相连，与 PE 一起共享主存储器；

• 每一个处理单元都拥有自己的控制单元 CU，各个控制器通过 CU 互联网络交换信息。

② 特点：系统资源管理易管理和利用：程序员无数据划分的负担，编程容易；系统处理机数目较少，不易扩充。

（2） 具有分布式存储器的多处理机

① 结构：具有分布式存储器的多处理机的基本结构如图 1.42 所示。

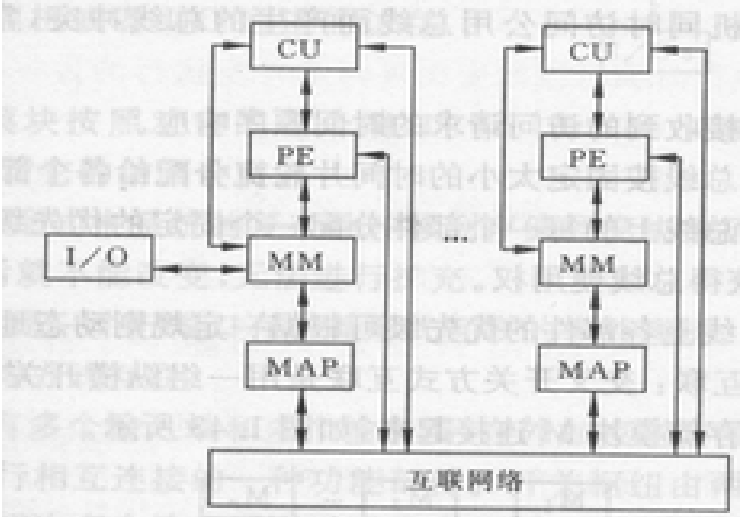


图 1.42 分布存储器多处理机基本结构

• 每一个处理单元都拥有自己的控制器 CU、本地存储器 MM，有的甚至带有 I/O 通道，从而形成相对独立的处理机模块；

• 这些模块通过互联网络进行相互通信。

② 特点：结构灵活；容易扩充；难以在各个处理单元之间实现复杂数据结构的数据传送；任务动态分配复杂；现有软件可继承性差，需要设计新的并行算法。

2.多处理机互联方式

多处理机互联方式对多处理机的性能有着不可低估的影响。要求能以较低成本实现各处理机间高速数据交换，同时提供灵活的连接机制以实现各种复杂甚至不规则的互联模式而不发生冲突。常用的连接方式有总线、交叉开头、多端口存储器和开头枢纽结构等。

（1） 总线互联：总线方式是多机处理机互联的一种最简单的结构形式。所有的功能部件(存储器、I/O 和处理机等)通过自各的接口与公用总线相连，各部件之间以分时或多路转换的形式进行信息传送。IBM stretch 和 UNIVAC larg 多处理机采用了单总线方式互联。单总线结构具有简单、成本低、系统上增减模块方便等优点。但是当任何一个总线接口线路出现故障时都会造成系统完全失效，而且当需要扩充处理机或存储器时会增加总线冲突而导致系统效率急剧下降。系统的总线的带宽和速度的限制。

常用两种方法来提高总结互联方式的性能：

① 使用高频同轴电缆来提高总线的传输速率。

② 使用多总线方式来减少访问总线的冲突概率，如西门子公司的结构式对称多处理机 SMS 采用了 8 总线互联。

为了解决多个处理机同时访问公用总线而产生的总线冲突，常用的总线仲裁算法如下：

①先来先服务：按接收到访问请求的时间顺顺序序响应。

②固定时间片：把总线按固定大小的时间片轮流分配给和个部件使用。

③静态优先级：为总线上的每一个部件分配一个固定的优先级，当多个部件争用总线时，优先级高的部件获得总线使用权。

④动态优先级：总线上各部件的优先级可根据一定规则动态地改变。

（2）交叉开关方式互联：交叉开关方式互联是用一组纵横开关阵列把横向的处理机动 P 和 I/0 通道与纵向的存储模块 M 连接起来，如图 1.43 所示。

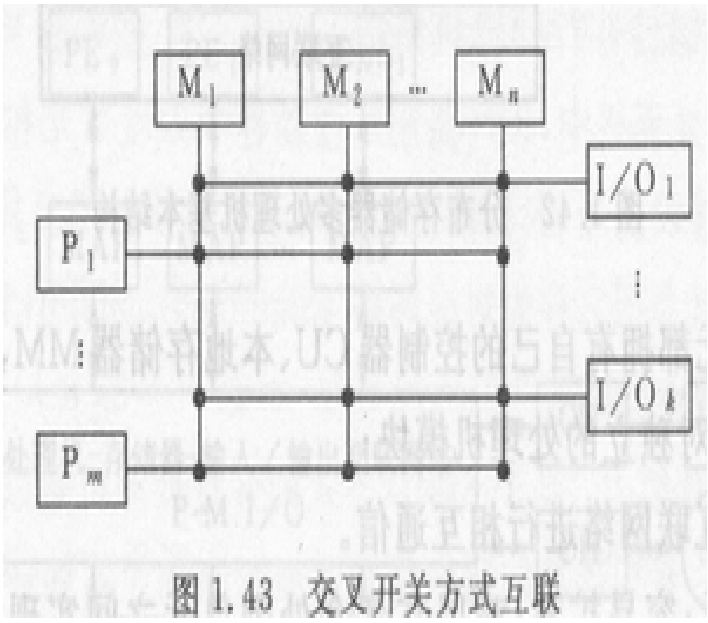


图 1.43 交叉开关方式互联

①存储器模块的全连接：交叉开头是多总线方式下总线数量增加的极端情况，每一个存储器模块都有自己的总线，总线数目 B=n+m+k 并且 n≥m+k，使 m 个处理机和 k 个 I/O 都能分到一套总线与 n 个存储模块中的某一个相连，大大展宽了互联传输的频带动提高了系统效率。

②复杂庞大的硬件结构：与公用总级系统采用按时间分配机制不同，交叉开关采用了按空间分配机制。它支持所有存储器模块同时传输，但是它的每个交叉点都必须具有多路转接逻辑功能要以实现切换并行传输，而且还需要具有解决访存冲突的能力。如此一来，整个交叉开关陈列是非常复杂的。

美国 C.mnp、S-1，Burroughs 的军用 D-825 及商用 B5500，B7700 等都采用了交叉开关形式互联。

（3） 多端口存储器方式互联：如果每个存储器模块有多个存取端口，且把分布在交叉开关网络中的控制、转接和优先仲裁逻辑分布到相应的存储器模块的接口，就形成了多端口存储器方式的结构，如图 1.44 所示。

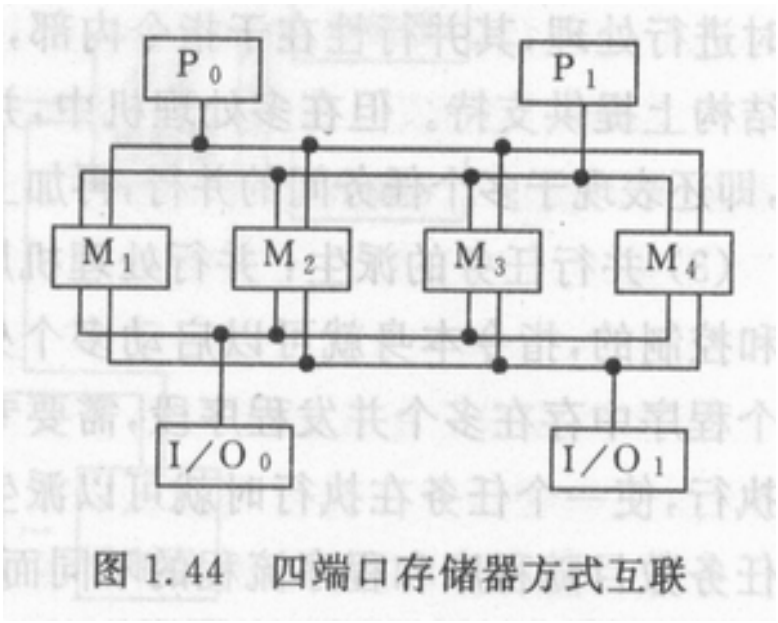


图 1.44 四端口存储器方式互联

① 核心是多端口存储器模块；

② 每个存储器模块的相应端口连接在一起，负责响应一个处理机或者 I/O 通道的访存请求；

③ 每个存储器模块按照对它的各个端口指定的优先级来分解对它的访问冲突；

④ 多端口存储器形式中系统的复杂性转移到存储器模块中了，因此其端口数不宜太多，而且一经制造以后就不能改变，无法进行扩充。
UNIVAC1108 就是一个五端口存储器形式互联的多处理机。

（4）开关枢纽结构方式互联

① 开关枢纽：具有多个输入端和多个输出端，在外加控制信号的作用下，输入端和输出端可以有选择地进行相互连接的一种功能部件。开关枢纽由两部分组成：仲裁单元和开关单元。仲裁单元解决多个输入端同时要求连到同一个输出端的冲突，开关单元把每个输入端近控制代码分别连接到相应的输出端。

②利用开关枢纽可以把每台处理机与周围邻近的处理机连接起来，组成各种具有分布结构的多处理机。

③理想的拓扑结构应该满足：所用的开关枢纽少；每个开关枢纽的端口数不宜太多；能以较短的路径把大量处理机连接起来实现快速灵活的相互通信；不改变模块本身结构就可以任意扩充系统规模。

美国加州大学伯克利分校的树形多处理机 X-TREE 就采用了开关枢纽结构方式进行互联。

3. 多处理要特点

多处理机属于 MIMD 系统，与 SIMD 的并行处理机相比，有很大的差别。其根源就在于两者的并行性等层次不同，多处理机要实现的是更高一层的作业任务间的并行。

下面通过与并行处理机进行比较来说明多处理机的主要特点。

（1） 结构灵活性：因为并行处理机主要针对数组、向量处理进行算法设计，具有一定专用性。尽管处理单元数多，但只需设置有限、固定的机间互联通路就能满足一批并行性高的算法的需求。而多处理机则有较强的通用性，机间互联模式相当复杂，以适应多种和算法，所以与并行处理机相比，多处理机中的处理机数还不算多。

（2） 程序并行性：并行处理机实现的是操作一级的并行，用一条指令即可对整个数组同时进行处理，其并行性在于指令内部，所以并行性的识别较容易，可以从指令类型及硬件结构上提供支持。但在多处理机中，并行性不仅表现在指令内部，而且还存在于指令外部，即还表现于多个任务间的并行，再加上系统通用性的要求，使得程序并行性的识别较难。

（3） 并行性任务的派生：并行处理机属于 SIMD，其并行操作是通过各条单独的指令反映和控制的，指令本身就可以启动多个处理单元的并行工作。多处理机属于 MIMD 系统，一个程序中存在多个并发程序段，需要专门的指令来表示这们的并发关系和控制它们并发执行，使一个任务在执行时就可以派生出可与之并发执行的另一些任务。派生出的并行任务数目随程序和程序流程的不同而变化。

（4） 进程同步：并行处理机实现的是指令内部对数据操作的并行，所有活跃的处理单元同时执行同一条指令的操作，受同一个控制器控制，工作自然是同步的。而多处理机实现的是指令、任务、作业级的并行，一般而言，同一时刻不同处理机执行着不同的指令，不可能也不需同步。但是，如果并发程序段之间有数据往来或控制依赖，就需采取专门的同步措施，以保证程序正确执行。

4. MPP 与 SMP

（1） 大规模并行处理机（MPP）：MPP 是随着微处理器技术的飞跃和应用需求的不断攀升在 20 世纪 80 年代中期发展起来的一种新机型。它是由众多的微处理器（从几百到上万）组成的大规模的并行系统。MPP 的出现成为计算机领域中一个研发热点，被用作开发万亿次甚至更高速的巨型机的主要结构。MPP 可以采用市场上的出售的 RISC 处理器，所以有很高的性价比。其结构如图 1.45 所示。

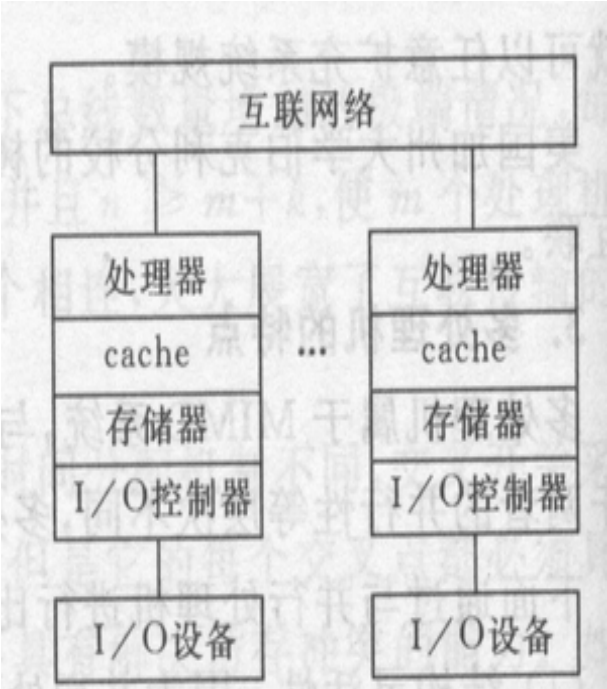


图 1.45 MPP 的结构

（2） 对称多处理机（SMP）：其结构如图 1.46 所示，SMP 目前也基于 RISC 微处理器。它与 MPP 最大的差别在于存储系统：SMP 有一个统一的共享主存空间，而 MPP 则是每个微处理器都拥有自己的本地存储器.

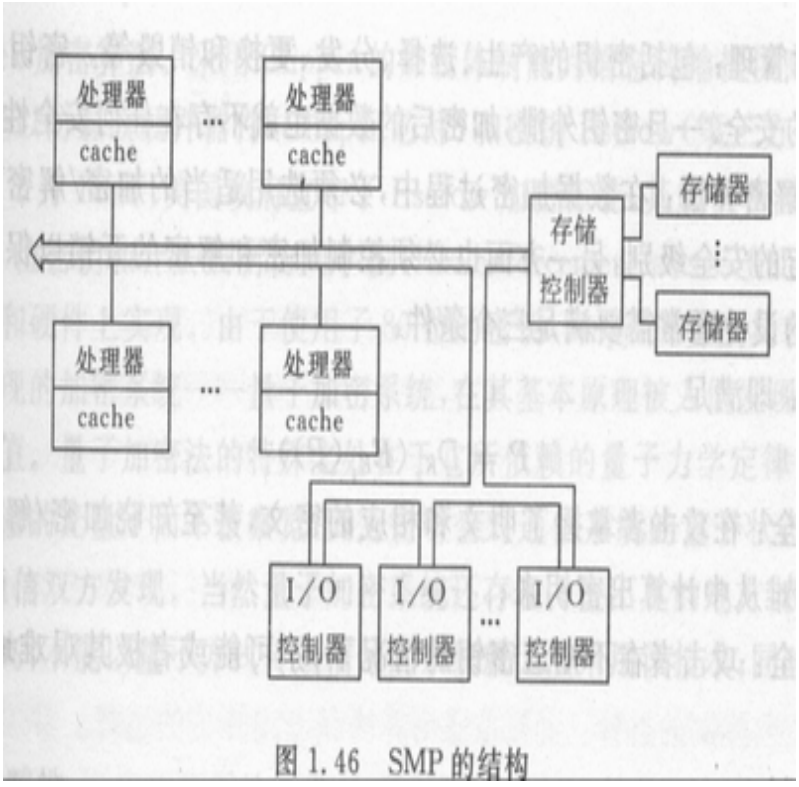


图 1.46 SMP 的结构

7 计算机安全性、可靠性及性能评价初步

1.7.1 数据的安全与保密

计算机的安全问题日益引起国际上的广泛关注，从 20 世纪 80 年代起，国际信息处理协会（IFIP）就开始进行信息处理系统的安全和保护方面的研究和探讨。国际标准化委员会对计算机安全的定义提出如下建议：″为数据处理系统建立和采取的技术的和管理的安全保护措施，保护计算机硬件、软件、数据不因偶然的事恶意的原因而遭破坏、更改和泄露。″数据加密是计算机中最重要的技术措施之一。

1. 数据加密和解密

数据加密即是对明文（未经加密的数据）按照某种的加密算法（数据的变换算法）进行处理，而形成难以理解的密文（经加密后的数据）。即使是密文被截获，截获方也无法或难以解码，从而防止泄露信息。

数据加密和数据解密是一对可逆的过程,数据加密是用加密算法 E 和加密密钥 K_1 将明文 P 变换成密文 C，表示为：

C= E_{K_1} （P）

数据解密是数据加密的逆过程，用解密算法 D 和解密密钥 K2，将密文 C 转换在明文 P，表示为：

P= D_{K_2} （C）

数据加密技术的关键在于：

（1） 密钥的管理：包括密钥的产生、选择、分发、更换和销毁等。密钥的安全程度直接影响着数据的安全，一旦密钥外泄。加密后的数据也就不存在任何安全性可言了。

（2） 加密/解密算法：在数据加密过程中，必须选用适当的加密/解密算法。一方面要保证达到给定的安全级别，另一方面也必须控制加密和解密的开销以保证性价比。加密和解密算法的设计通常需要满足三个条件：

① 可逆性：即满足

$$P= D_{K_2}(E_{K_1}(P))$$

②密钥安全：在攻击者掌握了明文和相应的密文，甚至知晓加密/解密算法的情况下，无法或者很难从中计算出密钥来；

③数据安全：攻击者在不知道密钥的情况下，不可能或者极其艰难地根据密文，推算出明文来。

2. 密钥体制

按照加密密钥 K_1 和解密密钥 K_2 的异同，有两种密钥体制：

（1） 秘密密钥加密体制（$K_1 = K_2$）

加密和解密采用相同的密钥，因而又称为对称密码体制。因为其加密速度快，通常用来加密大批量的数据。典型的方法有日本 NTT 公司的快速数据加密标准（FEAL）、瑞士的国际数据加密算法（IDEA）和美国的数据加密标准（DES）。

DES（数据加密标准）是国际标准化组织（ISO）核准的一种加密算法，自 1976 年公布以来得到广泛的应用，但近年来对它的安全性提出了疑问。1986 年美国政府宣布不再支持 DES 作为美国家数据加密标准，但同时又不准公布用来代替 DES 的加密算法。其他各国也有自己的商业利益和安全考虑。所以 ISO 标准中核准的加密算法至今未见有大的变化。由于 DES 算法本身存在的固有弱点以及人们已有对它进行攻击的办法，发达国家都在考虑新的算法以替代 DES。

（2） 公开密钥加密体制（$K_1 \neq K_2$）

又称不对称密码体制，其加密和解密使用不同的密钥；其中一个密钥是公开的，另一个密钥保密的。由于加密速度较慢，所以往往用在少量数据的通信中。典型的公开密钥是保密的。由于加密度较慢，所以往往用在少量数据的通信中。典型的公开密钥加密方法有 RSA 和 NTT 的 ESIGN。

RSA 算法的保密性取决于数学上将一个大数分解为两个素数的问题的难度，根据已有的数学方法，其计算量极大，破解很难。但是加密/解密时要进行大指数模运算，因此加密/解密速度很慢，影响推广使用。

一般 DES 算法的密钥长度为 56 位，RSA 算法的密角长度为 512 位。为了加速 DES 算法和 RSA 算法的执行过程，可以用硬件电路来实现加密和解密。

1993 年 4 月 16 日，美国政府推出了 Clipper 密码芯片，该芯片采用美国国家安全局设计的 Skipjack 加密算法。采用 Clipper 的加密体制能为信息传输提供高等级的安全和保密，该体制是以防篡改硬件器件（Clipper 芯片）和密钥 Escrow（第三方托管）系统为基础的。1994 年 2 月 14 日，美国政府宣布了 Escrow 加

密标准，其加密算法使用 Skipjack。该算法采用 80 位密钥和合法强制访问字段（law enforcement access field ，LEAF），以便在防篡改芯片和硬件上实现。由于使用了 80 位的密钥，Skipjack 算法具有较高的强度。

一种新出现的加密系统--量子加密系统，在其基本原理被人们发现十几年后，终于展现了它的价值。量子加密法的特殊之处在于它所依赖的量子办学定律，依据现有的技术无法窃听传输的光量子加密系统的量子状态造成不可逆的影响，从而被通信双方发现。当然量子加密系统还存在一些不足的地方，例如它对传输系统的要求比较高，不能以量子方式传送信号的系统就无法采用该技术；同时它也不支持技术数字签名。

3. 数据完整性

数据完整性保护是在数据中加入一定的冗余信息，从而能发现对数据的任何修改、增加或删除。通常采用的完整性检查方法是在发送或写入时对所有保护的数据的"检验和"作加密运算，产生报文验证码（MAC），附在数据后面。在接收或读出时，根据约定的的密钥对数据的检验和作加密运算，将所得结果与 MAC 比较，根据是否一致就可确定数据的完整性，MAC 码的长度要根据应用确定，一般在 16 位到 128 位之间。

在某些商业或金融领域内，由于其行业要求，需要防止通信的一方否认或伪造通信内容，这时通常采用"数字签名"的方法。

一般来讲，数字签名的目的就是在保证真实的发送与真实的接收方之间传送真实的信息。因而完美的签名机制应体现发送方签名发送，接收方签名送回执。数字签名不同于传统的书面签名。数字签名是在一定的密码算法下，由密钥和数据参与运算的结果，因此它有两个特点：一是动态变化，随着密钥和数据的不同而不同，而传统的人工签名是不变的；二是签名与数据不可分离。数字签名利用非对称的密钥体制，其特点是有一个公钥 K_p 和一个密钥 K_s；发方本人掌握密钥 K_s，而将公钥 K_p 公开；在数字签名时，发方先用密钥 K_s 对其文件检验和签名：

$$M = D_{K_s}(P)$$

然后接收方用公钥 K_p 对其签名验证：

$$P = E_{K_p}(M)$$

由于 K_s 只有签名者自己掌握，因此其他人无法伪造签，篡改数据。一旦签名验证符合，发送方无法否认曾经签发的数据。采用类似的方法，使接收方也无法否认曾经签发的回执。

1985 年,MIT 的 Goldwasser 和 Micali 及多伦多大学的 Rackoff 提出了零知识证明概念，指的是在没有暴露确凿信息内容的情况下能得到对方用户的承认方法。随着零知识证明理论的发展，它将与数字签名在鉴别技术中发挥更大的作用。

4. 密钥管理

数据加密的安全性在很大程度上取决于密钥的安全性。密钥管理包括密钥密码体制的选择、密钥的分发、现场密钥保护以及密钥的销毁。在任何时候密钥的泄露都会带来难以预料的损失。

（1） 密钥密码体制的选择：主要通用算法和专用算法的选择。常用的通用算法有 DES 算法和 RSA 算法。另外在某个领域或某个应用内部可能采用专用算法，专用算法本身是保密的，而且由于其应用范围有局限性，因此很少有攻击者愿意付出巨大的代价去破译。但专用算法没有经过深入研究和评审，其安全可信度可能会存在问题，所以使用时应慎重。

（2） 密钥的分发：需要防止在传送密钥的过程中发生泄露。通常实际使用的密钥至少有两个以上的层次，高层次的密钥称为主密钥。主密钥的传送要求绝对可靠，可以作为保密邮件，由专人专递，也可将密钥分成几部分分别传送。

（3） 密钥的保护：装有密钥的设备必须能够防侵害或者攻击。例如采取独特的封装技术,使得打开该设备会导致结构的破坏,以防止对设备内容的窃取。

（4） 密钥的销毁：所有的密钥都应有限定的使用周期，超过使用期限的密钥应销毁。销毁措施也要绝对可靠，应销毁包括纸张在内的所有存储介质上的密钥。

5. 磁介质上的数据加密

通常采用磁介质存储器（软盘、硬盘等）来保存和传递数据。所以有必要对其采取保护措施。常用的方法有：硬加的防复制技术、软加密的防解读技术和防跟踪技术。

（1） 硬加密技术：通常采用以下三种方式

①利用非标准格式的磁介质记录方式：例如使用非标准的扇区标识数据，使用非标准的磁记录编码方式,使用非标准的磁盘格式化规则以及使用非常规的数据区地址标志等。

②激光加密技术：利用精密定位技术与激光技术，在磁盘上某个位置产生永久性的标志。这种磁盘的复制很困难。

③利用专用的硬件：磁介质上的数据必须与专用的硬件配合在一起才能使用。

（2） 防跟踪技术:通过改变计算机系统的一些功能,使得跟踪、反汇编、设置断点、读取内存等操作无法执行。

1.7.2 计算机系统的可靠性

1. 计算机可靠性概述

计算机系统的硬件故障通常是由元器件的失效引起的。对元器件进行寿命试验并根据实际资料统计得知，元器件的可靠性可分成三个阶段，开始阶段器件工作处于不稳定期，失效较高；第二阶段器件进入正常工作期，失效率最低，基本保持常数；第三阶段元器件开始老化，失效率又重新提高；这就是所谓的"浴盆曲线"。因此应保证在计算机中使用的元器件处于第二价段。在第一阶段对元器件应进行老化筛选，而到了第三个阶段，则淘汰该计算机。

计算机系统的可靠性是指从它开始运行（t=0）到某时刻 t 这段时间内能正常运行的概率，用 R(t)表示。所谓失效率是指单位时间内失效的元件总数的比例，以 λ 表示，当 λ 为常数时，可靠性与失效率的关系为：

$R(t)=e^{-\lambda t}$

两次故障之间系统能政常工作的时间的平均值称为平均无故障时间（MTBF）：

$MTBF=1/\lambda$

通常用平均修复时间（MTRF）来表示计算机的可维修性，即计算机的维修效率，指从故障发生到机器修复平均所需要的时间。计算机的可用性是计算机的使用效率，它以系统在执行任务的任意时刻能正常工作的概率 A 来表示：

$A=MTBF/(MTBF+MTRF)$

计算机的 RAS 技术，就是指用可靠性 R、可用性 A 和可维修性 S 三个指标衡量一个计算机系统。但在实际应用中，引起计算机故障的原因除了元器件以外还与组装工艺逻辑设计等因素有关。因此不同厂家生产的兼容机，即使采用相同的元器件，其可靠性及 MTBF 也可能相差不大。

2. 计算机可靠性模型

计算机系统是一个复杂的系统，而且影响其可靠性的因素也非常繁复，很难直接对其进行可靠性分析;但通过建立适当的数学模型,把大系统分割成若干子系统，可以简化其分析过程。常见的系统可靠性数学模型有以下 3 种：

（1）串联系统：假设一个系统由 N 个子系统组成当且仅当所有的子系统都能正常工作时，系统才能正常工作，这种系统称为串联系统，如图 1.47 所示。设系统各个子系统的可靠性分别用 R_1 ， R_2,…， R_N 来表示,则系统的可靠性 R 可由下式求得：

R= R_1 R_2…… R_N

如果系统的各个子系统的失效率分别用 λ_1 ， λ_2,…， λ_N 来表示,则系统的失效率 λ 可由下式求得：

λ = λ_1 + λ_2 +…… λ_N

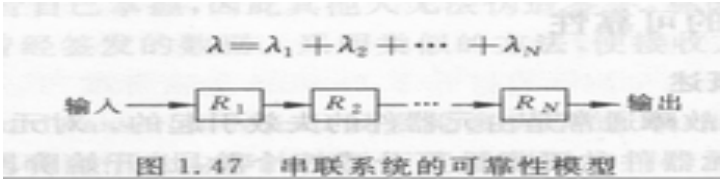


图 1.47 串联系统的可靠性模型

【例 1.2】设计算机系统由 CPU、存储器、I/O 三部分组成，其可靠性人别为 0.95 和 0.91 和 0.98.求计算机系统的可靠性.

解:R= R_1 • R_2 • R_3 =0.95.×0.90×0.85×0.73

计算机系统的可靠性为 0.73.

(2) 并联系统:假如一个系统由 N 个子系统组成,只要有一个子系统,如图 1.48 所示.设每个子系统的可靠性分别以 R_1 R_2…… R_N 表示,整个系统的可靠性 R 可由下式来求得: λ

R=1-(1- R_1)(1- R_2) …(1- R_N)

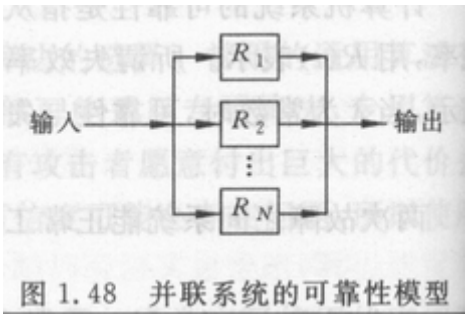


图 1.48 并联系统的可靠性模型

假如所有的子系统的失效率均为 λ,则系统的失效率为 μ:

$$\mu = \frac{1}{\frac{1}{\lambda}\sum_{j=1}^{N}\frac{1}{j}}$$

在并联系统中只有一个子系统是真正需要的,其余 N-1 个子系统称为冗余子系统,随着冗余子系统数量的增加,系统的平均无故障时间也增加了.

【例 1.3】设一个系统由三个相同子系统构成,其可靠性为 0.9,平均无故障时间为 10000 小时，求系统的可靠性和平均无故障时间.

解: $R_1 = R_2 = R_3 = 0.9$ $\lambda_1 = \lambda_2 = \lambda_3 = 1/1000 = 1\times 10^{-4}$（小时）

系统可靠性 $R=1-(1-R_1)^3=0.999$

系统平均无故障时间 MTBF 为:

$$\text{MTBF}=\frac{1}{\mu}=\frac{1}{\lambda}\sum_{j=1}^{3}\frac{1}{j}=\frac{1}{\lambda}\times\left(1+\frac{1}{2}+\frac{1}{3}\right)=18333 \text{（小时）}$$

(3) N 模冗余系统:N 模冗余系统由 N 个(N=2n+1)相同的子系统和一个表决器组成,表决器 N 个子系统中占我数相同结果的输出作为系统的输出,如图 1.49 所示.

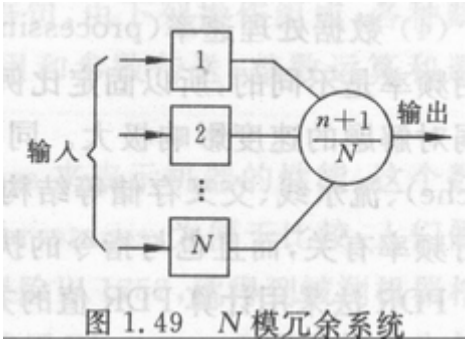


图 1.49 N 模冗余系统

整理人：刘海云
整理时间：2003.05

在N个子系统中,只有 n+1 个或n+1 个以上子系统能正常工作,系统就能正常工作,输出正确结果.假设表决器是完全可靠的,每个子系统的可靠性为 R_0 ,则N模冗余系统的可靠性为:

$$R = \sum_{i=n+1}^{N} \binom{i}{N} \times R_0^i (1-R_0)^{N-i}$$

其中, $\binom{i}{N}$ 表示从N个元素中取 i 个元素的组合数.

提高计算机的可靠性一般采取两项措施:

• 提高元器性质量,改进加工工艺与工艺结构,完善电路设计;

• 发展容错技术,使得在计算机硬件有故障的情况下,计算机仍能继续运行,得出正确的结果.

1.7.3 计算机系统的性能评价

无论是生产计算机的厂商还是使用计算机的用户都需要有某种方法来衡量计算机的性能,作为设计、生产、购买和使用的依据。但是由于计算机系统是一个极复杂的系统，其体系结构、组成各实现都有若干种策略；而且其应用领域也千差万别，所以很难找到统一的规则或标准去评测所有的计算机。

1. 性能评测的常用方法

（1） 时钟频率：计算机的时钟频率在一定程度上反映了机器速度，一般来讲，主频越高，速度越快。但是相同频率、不同体系结构的机器，其速度可能会相差很多倍，因此还需要用其他方法来测定机器性能。

（2） 指令执行速度：在计算机发展的初期，曾用加法指令的运算速度来衡量计算机的速度，速度是计算机的主要性能指标之一。因为加法指令的运算速度大体上可反映出乘法、除法等其他算术运算的速度，而且逻辑运算、转移指令等简单指令的执行时间往往设计成与加法指令相同,因此加法指令的运算速度有一定代表性。当时表征机器运算速度的单位是KIPS（每秒千条指令），后来随着机器运算速度的提高，计量单位由KIPS发展到MIPS（每秒百万条指令）。

（3） 等效指令速度法：随着计算机指令系统的发展，指令的种类大大增加，用单种指令的MIPS值来表征机器的运算速度的局限性日益暴露，因此很快就出现了改进的办法，称之为吉普森（Gibson）混合法或等效指令速度法。

等效指令速度法统计各类指令在程序中所占比例，并进行折算。设某种指令 在程序中所占比例为 w_i ，执行时间为 t_i，则等效指令的执行时间为：

$$T=\sum_{i=1}^{n}(w_i \times t_i)$$

其中 n 为指令的种类数。

（4） 数据处理速率（processing data rate ,PDR）法：因为在不同程序中，各类指令使用频率是不同的，所以固定比例方法存在着很大的局限性；而且数据长度与指令功能的强弱对解题的速度影响极大。同时这种方法也不能反映现代计算机中高速缓冲存储器（cache）、流水线、交叉存储等结构的影响。具有这种结构的计算机的性能不仅与指令的执行频率有关，而且也与指令的执行顺序与地址的分布有关。

PDR 法采用计算 PDR 值的方法来衡量机器性能，PDR 值越大，机器性能越好。PDR 与每条指令和每个操作数的平均位数以及每条指令的平均运算速度有关，其计算方法如下：

PDR=L/R

其中：L=0.85G+0.15H+0.4J+0.15K

R=0.85M+0.09N+0.06P

式中：G 是第条浮点指令的位数， M 是平均定点加法时间，

H 是每条打浮点指令的位数， N 是平均浮点加法时间，

J 是定点操作数的位数， P 是平均浮点加法时间，

K 是浮点操作的位数。

此外，还作了如下规定：G>20 位,H>30 位:从主存取一条指令的时间等于取一个字的时间;指令与操作存放在主存,无变址或间址操作;允许有并行或先行取指令功能,此时选择平均取指令时间.PDR 值主要对 CPU 和主存储器的速度进行度量,但不适合衡量机器的整体速度,因为它没有涉及 cache、多功能部件等技术对性能的影响。

（5） 核心程序法：上述性能评价方法主要是针对 CPU（有时包括主存）它没有考虑诸如 I/O 结构、操作系统、编译程序的效率等系统性能的影响，因此难以准确评价计算机的实际工作能力。

核心程序法是研究较多的一种方法，它把应用程序中用得最频繁的那部分核心程序作为评价计算机性能的标准程序，在不同的机器上运行，测得其执行时间，作为核心程序各部分之间的联系较小。由于程序短，所以访问存储器的局部性特征很明显，以 cache 的命中率比一般程序高。

2. 基准测试程序

基准程序法（benchmark）是目前一致承认的测试性能的较好方法，有多种多样的基准程序，如主要测试整数性能的基准程序、测试浮点性能的基准程序等。

（1） 整数测试程序：Dhrystone 是一个综合性的基准测试程序，它是为了测试编译器和 CPU 处理整数指令和控制功能的有效性，人为地选择一些"典型指令"综合起来形成的测试程序。
用 C 语言编写的 Dhrystone 基准程序用了 100 条语句，由下列操作组成：各种赋值语句；各种数据类型和数据区；各种控制语句；过程调用和参数传送；整数运算和逻辑操作。

Dhrystone 程序测试的结果由每秒多少个 Dhrystones 来表示机器的性能，这个数值越大性能越好。VAX11/780 的测试结果为每秒 1757Dhrystones,为便于比较，人们假设 1VAX MIPS =每秒 1757Dhrystones,将被测试机器的结果除以 1757，就得到被测试机器相对 VAX11/780 的 MIPS 值。有些厂家在宣布机器性能时就用 DhrystoneMIPS 值作为各自机器的 MIPS 值。

不过不同的厂家在测试 MIPS 值使用的基准程序一般是不一样的，因此不同厂家机器的 MIPS 值有时虽然是相同的，但其性能却可能差别很大，那是因为各厂家在设计计算机时针对不同的应用领域：如科学和工程应用、商业管理应用、图形处理应用等，而采用了不同的体系结构和实现方法。同一厂家的机器，采取相同的体系结构，用相同的基准程序测试，得到的 MIPS 值越大，一般说明机器速度越快。

（2） 浮点测试程序：在计算机科学和工程应用领域内，浮点计算工作量占大比例，因此机器的浮点加法、浮点乘法时间。而大部份工作则标出用 Linpack 和 Whetstone 基准程序测得的浮点性能。Linpack 主要测试向量性能和高速缓存性能。Whetstone 是一个综合性测试程序，除测试浮点操作规程外，还测试整数计算和功能调用等性能。

① 理论峰值浮点速度：巨型机和小巨型机在说明书中经常给出"理论峰值速度"的 MFLOPS 值。它不是机器实际执行程序时的速度，而是机器在理论上最大能完成的浮点处理速度。它不仅与处理机时钟周期有关，而且还与一个处理机里能并行执行操作的流水线功能部件数目和处理机的数目有关。多个 CPU 机器的峰值速度是单个 CPU 的峰值速度与 CPU 个数的乘积。

② Linpack 基准测试程序：linpack 基准程序是一个用 FORTRAN 语言写成的子程序软件包，称为基本线性代数子程序包，此程序完成的主要操作是浮点加法和浮点乘法操作。测量计算机系统的 Linpack 性能时，让机器运行 Linpack 程序，测

量运行时间，将结果用 MFLOPS 表示。
当解 n 阶线性代数方程组时，n 越大，向量化程度越高。其关系如表 1.2 所示.

表 1.2 矩阵的向量化程度

矩阵规模	100×100	300×300	1000×1000
向量化百分比	80%	95%	98%

向量化百分比指的是含向量成分的计算量占整个程序计算量的百分比.在同一台机器中,向量化程度越高,机器的运算速度越快,因为不管 n 大小,求解议程时花在非向量操作的时间差不多是相等的.

③Whetstone 基准测试程序:Whetstone 是用 FORTRAN 语言编写的综合性测试程序,主要由执行浮点运算、功能调用、数组变址、条件转移和超越函数的程序组成。Whetstone 的测试结果用 Kwips 表示，1Kwips 表示机器每秒钟能执行 1000 条 Whetstone 指令。

（3） SPEC 基准程序（SPEC benchmark）:SPEC 是 System Peformance Evaluation Cooperative 的缩写,是由几十家世界知名计算机大厂商所支持的非盈利的合作组织，旨在开发共同认可的标准基准程序。

SPEC 基准程序是由 SPEC 开发的一组用于计算机性能综合评价的程序。以对 VAX×11/780 机的测试结果作为基数，其他计算机的测试结果以相对于这个基数的比率来表示。SPEC 基准能较全面地反映机器性能，有很高的参与价值。

SPEC 版本 1.0 是 1989 年 10 月宣布的，是一套复杂的基准程序集，主要用于测量与工作和科学应用有关的数字密集型的整数和浮点数方面的计算。源程序超过 15 万行，包含 10 个测试程序，使用的数据量比较大，分别测试应用的各个方面。

SPEC 基准程序测试结果一般以 SPECmark（SPEC 分数）、SPECint（SPEC 整数）和 SPECfp（SPEC 浮点数）来表示。其中 SPEC 分数是 10 个程序的几何平均值，SPEC 整数是 4 个整数程序的几何平均值，SPEC 浮点是 6 个浮点程序的几何平均值。

1992 年在原来 SPECint89 和 SPECfp89 的基础上又增加了两个整数测试程序和 8 个浮点数测试程序，因此 SPECint92 由 6 个程序组成，SPECfp92 由 14 个程序组成。这 20 个基准程序是基于不同的应用写成的，主要测量 32 位 CPU、主存储器、编译器和操作系统的性能。

已有大约 30 个计算机软件和硬件公司参加了这个组织，它们都承认这种测试，并同意作为他们生产的机器或系统的测试标准。SPEC 基准程序的测试结果获得了普遍的认可。

参加这个组织的主要成员有：IBM，AT&T，BULL，Compag，CDC，DG，DEC，Fujitsu，HP，Intel，MIPS，Motorola，SGI，SUN 和 Unisys 等。1995 年，这些厂商又共同推出了 SPECint95 和 SPECfp95 作为最新的测试标准程序。

（4） TPC 基准程序：TPC 是 Transaction Processing council（事务处理委员会）的缩写，TPC 基准程序是由 TPC 开发的评价计算机事务处理性能的测试程序，用以评测计算机在事务处理、数据库处理、企业管理与决策支持系统等方面的性能，TPC 成立于 1998 年，目前已有 40 多个成员，几乎包括了所有主要的商用计算机系统厂商和数据库厂商。TPC 分别于 1989 年 10 月、1990 年 8 月和 1992 年 7 月发表了 TPC-A 和 TPC-B，TPC-C 三个基准测试程序规范，计划在其后发表的有 TPC-D 和 TPC-E。该基准程序的评测结果用每秒完成的事务处理数 TPC 来表示。TPC 基准测试程序在商业界范围内建立了用于衡量机器性能以及性能价格比的标准。但是任何一种测试程序都有一定的适用范围，TPC 也不例外。

1.7.4 计算机故障诊断与容错

1. 计算机故障诊断技术

（1） 计算机的故障：根据计算机故障表现出的特点，可以分为永久性、间歇性及瞬时性三类。

① 永久性的故障：永久性故障表现出稳定及持续性的特征，如元器件的损坏、电路的断线或短路、程序编写的错误等，它的特点是故障可以重复出现。

② 间隙性故障：间隙性故障表现出不稳定性及对系统状态具有信赖性的特征，此时可能表现出机器时好时坏的现象。

③ 瞬时性故障：瞬时性故障是由偶然原因引起的短暂故障，一般无需修复就能恢得正常。但若频繁出现，也要影响工作；所以需要查出故障原因，以消除影响。

无论何种故障，均需及时发现，采取措施，避免故障影响的扩散。故障可能源于硬件，也可能由软件引起。

（2） 故障诊断方法：故障诊断包括故障定位两个方面。

• 故障检测：测试并确定计算机系统有无故障的过程；

• 故障定位：判定故障发生在某个子系统、功能 块或器件的过程。

通常，故障诊断的主要方法有下述三种：

① 对电路直接进行测试的故障测试法：将被测试的系统划分成若干个测试域，并向这些域发送一系列测试码，然后收集并分析被测试区的返回码，以确定故障位置或找出产生故障的元器件。

这种按线路内部的电路结构逐个进行测试的方法，将随着电路规模的增大急骤地增加复杂性。因此对计算机系统往往是先采用下面提到的按功能测试的方法诊断到某一出故障的功能部件，然后再对这一部件的电路进行测试，或者干脆把这一产生故障的功能部件整个替换掉。

②"检查诊断程序"法：用机器语言写的"检查诊断程序"来进行诊断的方法是一种功能测试法。它利用机器指令功能来对系统某些部件进行测试。但由于一条指令的正确执行，往往涉及许多部件，因此故障定位所需的诊断时间较长；而且要求系统必须有能力保证诊断程序的正确执行，否则计算机连程序都不能运行，更谈不上诊断了。

③ 微诊断法：在微程序控制的计算机中用微指令对系统进行诊断叫做微诊断法。由微指令组成的微诊断程序存放在控制存储器中或者先存外存储器中，诊断法也是一种功能测试法。为了进行测试而必须保证工作只涉及较少部件，因此故障分辨得很细，诊断程序运行的时间也较短。

不论采用指令或微指令进行诊断都是采用滚雪球的方法。先对系统的某一部分进行测试，如果没有发现错误，就在这部分的基础上对其他未测试部分进行测试。如同滚雪球般，越滚越大，直到全部测试完成。

2. 计算机容错技术

容错是采用冗余方法来肖除故障影响。针对硬件，有时间冗余和元器件冗余两种方法

• 时间冗余：对同一计算进行重复运算，并对结果进行比较，或进行验算等，这种方法对解决偶然性故障比较有效。

• 元器件冗余：则利用附加的硬件来保证在局部有故障的情况下系统能正常工作。

容错的技术在不断进步，从最初的简单双机冗余到越来越高级的容错系统设计，保障了系统的可靠性和稳定性。

（1） 简单的双机备份：在 20 世纪 60 年代主要双处理机或双机的方法来达到容错的目的。例如把关键的元件（处理机、存储器等）或整个计算机设置两套：一是系统运行时使用，另一份用作备份。根据系统的工作情况又可分为：热备份和冷备份两种。

① 热备份（双重系统）：两套系统同时同步运行，当联机子系统检测到错误时，退出服务进行检修，而由"热备份"子系统接替工作。

② 冷备份（双工系统）：处于冷备份的子系统平时停机或者运行与联机系统无关的运算，当联机子系统产生故障时，人工或自动进行切换，使冷备份系统成为联机系统。在冷备份时，不能保证从程序端点处精确地连续工作，因为备份机不能取得原来的机器上当前运行的全部数据。

（2） 操作系统支持的双机容错：在 70 年代中期出现了软件和硬件结构的容错方法。该方法在操作系统的层次上，支持联机维修，即故障部分退出后运行、进行维修并重新投入运行都不影响正在运行的应用程序。该结构特点是系统内包括双处理器、双存储器、双输入输出控制器、不间断工作的电源以及与之适应的操作系统等。因此上述硬件的任一部分发生故障都不会影响系统的继续工作。系统容错是在操作系统控制下进行的，在每个处理机上都保持了反映所有系统资源状态的表格以及本机和它机的工作进程。

为了正常工作，必须保持双处理机工作同步，及时比较操作结果；并且提供检测机制以定位出故障的处理器（因为双机表决结果为 1：1，不能判定哪台机器出故障）

综上所述，对故障处理步骤及方法归纳如下：

① 故障封闭：防止故障扩散；

② 检错：利用冗余代码，提供出错信息，例如奇偶验码、海明检验码、CRC 校验码等；

③ 重执：利用软件重新执行运算，以清除瞬时异常的影响；

④ 诊断：对于非瞬时异常，确定故障性质及地点；

⑤ 系统重构及恢复：对故障部件从系统中退出，重建系统结构，然后恢复运行；

⑥ 修复：对故障部件进行在线修复或脱机修复，一般为后者；

⑦ 重入：故障部件修复后，重新投入系统运行。

第 1 章 计算机硬件基础知识

1.1 计算机系统概述

1.1.1 计算机史简介

自从 1946 年 2 月现代电子计算机的鼻祖 ENIAC(electronic numerical integrator and computer)在美国宾夕法尼亚大学问世后，短短 50 年里，计算机技术经历了巨大的变革。习惯上根据计算机系统所采用的硬件技术来划分计算机的发展阶段。

从 1946 年到 50 年代后期（1946-1957）为电子管计算机时期。计算机的元器伯主要由电子管（vacuum tube）组成。其特点是体积庞大、功耗高、运算速度较低。如 ENIAC 占地 170 ㎡，重达 30t。功耗为 140kW，有 18000 多个电子管，每秒钟能进行 5000 次加法计算。这一阶段，计算机主要用于军事、国防等尖端技术领域。除了 ENIAC 以外,1945 年左右,冯 o 诺依曼等人在研制 EDVAC(electronic discrete variable computer)时，提出了存储程序(stored-program)概念，奠定了以后计算机发展的基石。IBM 公司 1954 年 12 月推出的 IBM650 是第一代计算机的代表。

从 20 世纪 50 年代后期到 60 年代中期（1958-1964）为晶体管计算机时期。自从 1947 年晶体管（transistor）在贝尔实验室诞生后，引发了一场影响深远的电子革命。体积小、功耗低、价格便宜的晶体管取代了电子管，不仅提高了计算机的性能，也使计算机在科研、商业等领域内广泛地应用。第二代计算机不仅采用了晶体管管器件，而且存储器改用速度更快的磁芯存储器；与此同时高级编程语言和系统软件的出现，也大大提高了计算机的性能和拓宽了其应用领域。这一时期计算机的代表主要有 DEC 公司 1957 年推出的 PDP-I、IBM 公司于 1962 年推出的 7094 以及 CDC 公司 1964 年研制成功的 CDC6600。1969 年 CDC 公司研制的 CDC7600 平均速度达到每秒千万次浮点运算。

从 20 世纪 60 年代中期到 70 年代初期（1965-1971）为集成电路计算机时代。第一代和第二代计算机均采用分离器件（discrete component）组成。集成电路（integrated circuit）的出现，宣告了第三代计算机的来临。由于采用了集成电路,使得计算机的制造成本迅速下降;同时因为逻辑和存储器件集成化的封装，大提高了运行速度，功耗也随之下降；集成电路的使用，使得计算机内各部分的互联更加简单和可靠，计算机的体积出进一步缩小。这一时期的代表为 IBM 的 system/360 和 DEC 的 PDP-8。

从 20 世纪 70 年代初期到 70 年代后期（1972-1978）为大规模集成电路（LSI）计算机时代。20 世纪 70 年代初半导体存储器的出现。迅速取代了磁芯存储器，

计算机的存储器向大容量、高速度的方向飞速发展。存储器芯片从1kbit,4kbit,16kbit,64kbit,256kbit,1Mbit,4Mbit 发展到 16Mkbit（1992）年。

接着就进入了超大规模集成电路（VLSI）计算机时代。随着技术的日新月异，软件和通信的重要性也逐步上升，成为和硬件一样举足轻重的因素。同时系统结构的特点对计算机的性能也有巨大的影响（中断系统、Cache 存储器、流水线技术等等）。实际上在第三代计算机以后，就很难找到一个统一的标准进行划分。

目前更流行的是认为计算机的发展经过了三次浪潮（wave）。第一次浪潮为单个主机（Mainframe）的时期，IBM360、370 是其典型代表。第二次浪潮为客户/服务器（Client/Server）的时期，这时期出现了小型机、微型机和局域网。现在正处于第三次浪潮，网络计算的时期，即以网络为中心或以网络为基础的计算时期。

1.1.2 计算机系统的结构

计算机的工作过程一般是由用户（程序员）使用各种编程语言把所需要完成的任务以程序的形式提交给计算机，然后翻译成计算机能直接执行的机器语言程序，在计算机上运行以实现用户需要的功能。

计算机系统可以看作是一个多层的系统，不同的用户（或者应用）与不同的层次打交道。通常把计算机系统分作 4 层：物理机器（机器语言机器）、操作系统语言机器、汇编语言机器和高级语言机器，如图 1.1 所示。第 1 层实际上就是由硬件组成的计算机本身，它所能理解和执行的是机器语言。第 2～4 层其实都是"虚拟"机器，通过软件组成的层次结构，隐藏了机器的硬件实现细节。用户在其上见到的"机器"实际上是在物理机器上运行的软件表示的虚拟机器。

1945 年冯 o 诺依曼在设计 EDVAC 时提出了存储程序的概念，后来他和同事合作设计的存储程序计算机 IAS 的结构被其后几乎所有的通用计算机采用。这种结构也就称为冯 o 诺依曼机，如图 1.2 所示。其中主存（main memory）用以存放数据和程序；算术逻辑运算单元 ALU(arithmetic logical unit)完成二进制算术和逻辑运算；程序控制单元 PCU(program control unit)负责解释和执行指令；I/O(input/output)设备由 PCU 控制，实现和外部环境的数据交换。



图 1.1 计算机系统的层次结构

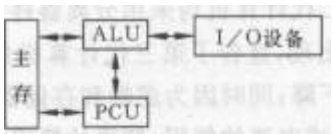


图 1.2 IAS 计算机结构

1.2 数据的计算机表示

1.2.1 数制

计算机的存储器（memory）和寄存器（register）是两态器件，因此，各种信息是以二进制的方式在计算机里存储和运算。但是为了方便起见，有时也需采用不同的数制（number system）来表示数据，例如十进制、十六制、八进制等。

1. 数制介绍

一种数制由基数 r(radix)和 r 个不同的数码(symbol)组成。任意一个数用 r 进制可以表示为

$N=(R_nR_{n-1}\cdots R_1R_0R_{-1}\cdots R_{-m})_r$
$=R_n\times r^n+ R_{n-1}\times r^{n-1}+\cdots+ R_1\times r+ R_0+ R_{-1}\times r^{-1}+\cdots+ R_{-m}\times r^n$

对于十进制而言共有 10 个数码：0，1，2，3，4，5，6，7，8，9 十进制数 538.4 即表示 $5\times10^2+3\times10^1+8\times10^0+4\times10^{-1}$。对于八进制数而言 8 个数码为 0～7；而十六进制的 16 个数码分别为 0～9 和 A～F。二进制的基数为 2,数码为 0 和 1 。

上式称为数 N 在 R 进制中的按权（power）展开式。

【例 1.1】把八进制数 1704.25 转化为十进制数。

解：
$(1704.25)_8=1\times8^3+7\times8^2+0\times8^1+4\times8^0+2\times8^{-1}+5\times8^{-2}$
$=1\times512+7\times64+4\times1+2\times0.125+5\times0.015625$
$=(964.328125)_{10}$

【例 1.2】把十六进制 2EC.F 转化为十进制数。

解：
$(2EC.F)_{16}=2\times16^2+14\times16^1+12\times16^0+15\times16^{-1}$
$=2\times256+14\times16+12\times1+15\times0.625$
$=(748.9375)_{10}$

表 1.1 列出了常用的十、十六、八和二进制的对照表。

表 1.1 常用进制对照表

十进制	十六进制	八进制	二进制	十进制	十六进制	八进制	二进制
0	0	0	0	9	9	11	1001
1	1	1	1	10	A	12	1010
2	2	2	10	11	B	13	1011
3	3	3	11	12	C	14	1100
4	4	4	100	13	D	15	1101
5	5	5	101	14	E	16	1110
6	6	6	110	15	F	17	1111
7	7	7	111	16	10	20	10000
8	8	10	1000				

2. 数制间的转换

（1）从十进制向 r 进制转换

把一个十进制数转换为等价的 r 进制数应分成两步进行：首先把该数的整数部分和小数部分分别转换为对应的 r 进制数；然后再把这两部分合起来即可。

算法 1：除 r 取余（把十进制数 I 转换为对应的 r 进制数）

① k：=1

② 把 I 除以 r 得到商 Q 和余数 Rk

③ 若 Q=0 则转到④，否则 I：=Q；k：= k+1，并转到②

④ 把 Rk 按 k 从大到小排列起来就是所求结果

【例 1.3】把（307）$_{10}$转换为对应的二进制数。

解：

商（Q）	……余数（Rk）
307	
153	1
76	1
38	0
19	0
9	1
4	1
2	0
1	0

0 1

（307）$_{10}$=（100110011）$_{2}$

【例 1.4】把（307）$_{10}$转换为八进制数。

解：

商（Q）……余数（Rk）
307
38 3
4 6
0 4

（307）$_{10}$=（463）$_{8}$

【例 1.5】把（307）$_{10}$转换为十六进制数。

解：

商（Q）……余数（Rk）
307
19 3
1 3
0 1

（307）$_{10}$=（133）$_{16}$

算法 2：乘 r 取整（把十进制小数 F 转换为 r 进制数）

①k：=1

②F 乘以 r 得到整数部分 Rk 和小数部分 F’

③若 F’=0 或者 k 达到规定的精度位数则转到④，否则 F：=F’；k：=k+1，并转至②

④把 Rk 按 k 从小到大排列起来，其中 R1 是最靠近小数点的位

【例 1.6】 把（0.8125）$_{10}$转换为二进制数。

解：

```
    0. 8125↵
      × 2↵
  ------------
1   . 6250↵
      × 2↵
  ------------
1   . 250↵
      × 2↵
  ------------
0   . 50↵
      × 2↵
  ------------
1   . 0↵
```

$$(0.8125)_{10}=(0.1101)_{2}$$

【例 1.7】把（0.8125）$_{10}$转换为十六进制数和八进制数。

解：

```
    0. 8125↵
      × 16↵
  ------------
D   . 0↵
```

```
    0. 8125↵
      × 8↵
  ------------
6   . 50↵
      × 8↵
  ------------
4   . 0↵
```

$$(0.8125)_{10}=(0.D)_{16}=(0.64)_{8}$$

（2）二进制数/八进制/十六进制数之间的相互转换

因为 $2^3=8$，$2^4=16$，所以每一位八进制数码都对应着一个 3 位二进制数，每一位十六进制数码对应着一个 4 位二进制数，如表 1.2 和表 1.3 所示。

表 1.2 八进制数和二进制数对应表

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

表 1.3 十六进制数和二进制数对应表

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111
8	9	A	B	C	D	E	F
1000	1001	1010	1011	1100	1101	1110	1111

二进制向八进制转换的方法为：从小数点开始，分别向左、右每 3 位二进制数编成一组，若不够 3 位，则小数点左侧的最高位和右侧的最低位用 0 补充；每一组用对应的八进制数码表示即可。八进制向二进制转换的方法为：从小数点开始，把每一位八进制数码转换成对应的 3 位二进制数即可。其小数点左侧的最高位或右侧的最低位的 0（如果有的话）可以省去。二进制和十六进制数之间的转换与此类似，只不过每组的二进制位数是 4 而不是 3。

【例 1.8】把（273.64）$_8$转换为二进制数。

解：

2	7	3	.	6	4
010	111	011	.	110	100

$(273.64)_{10}=(10111011.1101)_2$

【例 1.9】把$(7CE2.9A)_{16}$转换为二进制数。

解：

7	C	E	2	.	9	A
0111	1100	1110	0010		1001	1010

$(7CE2.9A)_{16}=(111110011100010.1001101)_2$

在计算机中八进制数、十六进制数通常被认为是二进制数的另一种表示方法。

（3）特殊方法

①由于八进制与二进制之间的转换十分简单，因此有时十进制与二进制这间的转换可以以八进制为中介。

（273.71875）$_{10}=(421.56)_8=(100010001.10111)_2$

②如果一个分数的分母为 2 的整数次幂则可以用以下方法转换。

$(3/32)_{10}=3\times2^{-5}=(11)_2\times(0.00001)_2=(0.00011)_2$

③从十进制向八进制/十六进制转换可以先把十进制转换为二进制，然后再从二进制转换为八进制/十交进制。

$(27.625)_{10}=(11011.101)_2=(33.5)_8=(1B.A)_{16}$

1.2.2 数值数据的表示方法

因为当前的计算机所采用的存储器件都是两态器件，所以适合于存放二进制数据。尽管为了方便起见，也使用其他进制，如十进制、十六进制，但是在计算机内部数据还是以二进制的形式存放和处理。二进制只使用了两个不同的数字符号，易于用物理器件来实现：在物理世界中具有两个稳定状态的物理器件很多，如晶体管制"截止"与"导通"、电容的"充电"与"放电"、电压信号的"高"与"低"、脉冲的"有"与"无"，电磁单元的"正向磁化"与"反向磁化"等等，只要规定其中一种稳定状态表示"1"，另一种稳定状态表示"0"，就可以用来表示二进制数位了。其次，二进制的运算规则非常简单，易于用电子器件来实现。

计算机里用到的数据类型主要可分为两类：表示数量的数值数据和非数值性的符号数据。所有的数据都是以二进制的形式在计算机里处理和存储。本节主要讲述数值数据在计算机里的表示方法。

1.符号位的处理
正数和 0 都可以表示为无符号数，而负数通常是用"-"号来表示。但在计算机里，任何符号都是用"1"和"0"来表示的。所以，习惯上计算机里都用最高位（最左位）来表示数的符号：0--正数，1--负数。

例如，X1=（+0010010）$_2$和 X2=（-0010010）$_2$在计算机里分别表示为 00010010 和 10010010。

2.小数点的处理

计算机在处理数值数据时，对小数点的处理有两种不同的方法：定点法和浮点法，也就对应了两种不同形式的数据表示方法。

（1）定点数据表示方法

在这种表示方法中，小数点的位置固定不变。根据其所在位置的不同，分别用来表示整数和纯小数：

• 定点整数格式：$N=b_s b_m b_{m-1}\cdots b_1 b_0$
小数点位置在最低数据位 b0 的右边，bs 为符号位，bm 为其最高数据位，其取值范围为｜N｜≤2^m-1,即$-(2^m-1)\leqslant N\leqslant(2^m-1)$。

• 定点小数格式：$N=b_s.b_{-1}b_{-2}\cdots b_{-n}$
小数点位置固定在符号位 b_s 与最高数据倦 b_{-1}之间。其取值范围｜N｜≤$1-2^{-n}$ 。

需要注意的是,无论是在定点整数还是定点小数表示法中,小数点作为一个符号并不明确地表示出来.而是由其表示方法所隐含。例如:

N=10111010，若表示定点小数则其值为-0.011101；若表示定点整数，则其值为-111010。

(2)浮点数据表示方法

这种表示方法把数值数据表示成

$N=M*R^E$

其中，M 被称为 N 的尾数，E 是 N 的指数或称阶码，而 R 则是该阶码的基数。在这种表示方法下，同一个数可以有不同的浮点表示形式，阶码的大小可以用来调节数值中小数点的位置。例如：0.081 可以表示为 81E-3，也可以表示为 8.1E-2 或者 0.81E-1 等等,这里 E 只是尾数和阶码之间的一个分隔符,而隐含基数为 10。在这种意义上，其小数点位置是可以"浮动"的，所以把该方法称为浮点数据表示法。

需要指出的是在我们上面的表示方法中，其基数 10 并未出现在表示形式中，而是隐含规定的。在计算机中，基数 R 通常由硬件部件（浮点运算部件）隐含规定，也不明确的出现在表达式中。在计算机里浮点数通常以如下形式表示：

Ms	E	M

Ms 为尾数 M 的符号位，其位数为 1；M 为用定点小数表示的尾数的绝对值，其位数 n 决定了该浮点数的精度；E 为用定点整数表示的阶码，其位数 m 则决定该浮点数的取值范围。

IEEE 规定了 3 种常用的浮点数格式：短浮点数格式，长浮点数格式以及临时浮点数格式，见表 1.4。临时浮点数主要是用来表示运算的中间结果，以保证运算的精度。

表 1.4 3 种常用的浮点数格式表

	符号位	阶码	尾数	总位数
短浮点数	1	8	23	32
长浮点数	1	11	52	64
临时浮点数	1	15	64	80

3. 定点数据编码方法

对于数值数据而言，无论以定点形式表示或是以浮点形式表示，都需要通过某种编码方法，以计算机所能处理的 0 和 1 二进制位来表示其数值大小。通常一个数值数据的机内表示形式称为其机器码，而一个机器码所代表的数值称为该机器码的真值。

我们假设机器码的宽度为 n，最高位为符号位，下面分别介绍实际应用中常见的 4 种编码方法：原码、反码、补码和移码（又称增码）。

（1）原码表示法

• 定点整数原码表示法

定点整数原码的定义如下所示：

$$[X]_{原}=\begin{cases} X & 0\leqslant X<2^{n-1} \\ 2^{n-1}-X & -2^{n-1}<X\leqslant 0 \end{cases}$$

由定义可以看出，正整数的原码就是其自身,而负整数的原码只需把其绝对值的原码的符号位置为 1 即可(0 表示正号,1 表示负号)。原码表示法的取值范围为

$-(2^{n-1}-1)\leqslant X\leqslant 2^{n-1}-1$。

注意在原码表示法中，零有正零和负零之分。[+0]=0000…0，[-0]=1000…0。

• 定点小数原码表示法

定点小数原码的定义如下所示：

$$[X]_{原}=\begin{cases} X & 0\leqslant X<1 \\ 1-X & -1<X\leqslant 0 \end{cases}$$

正的纯小数的原码就是其自身，而负的纯小数的原码可以通过把其绝对值的原码的符号位置 1 来得到。其取值范围为$-(1-2^{-(n-1)})\leqslant X\leqslant(1-2^{-(n-1)})$。

综上可以看出一个数的原码是由符号位加上数值位绝对值组成，符号位为 0 代表正数，符号位为 1 代表负数。由于 0 有+0 和-0 两种表示方法，n 位二进制数可以表示 2n －1 个原码。

用原码实现乘除运算规则较简单，但做加减运算不方便。原码表示法的一个主要优点在于其真值和编码表示之间对应关系很直观，容易转换。

（2）补码表示法

• 定点整数的补码表示

定点整数补码的定义如下所示：

$$[X]_{补}=\begin{cases} X & 0\leqslant X<2^{n-1} \\ 2^{n}+X & -2^{n-1}\leqslant X<0 \end{cases}$$

正整数的补码就是其自身，负整数的补码可以通过对其绝对值部分逐位求反，并在最低位加 1 求得。同样，在补码表示法中，符号位为 0 表示正号，为 1 表示负

号。其表示的真值 X 的范围为$-2^{n-1} \leqslant X \leqslant 2^{n-1}-1$。在补码表示中，0 的机器码是惟一的，没有+0 和-0 之分。

• 定点小数的补码表示

定点小数补码的定义如下所示：

$$[X]_{补}=\begin{cases} X & 0 \leqslant X < 1 \\ 2+X & -1 \leqslant X < 0 \end{cases}$$

定点小数的补码与定点整数的补码求法相似。其表示的真值 X 的范围为 $-1 \leqslant X \leqslant 1-2^{-(n-1)}$。当 n=8 时，-1 的定点小数补码表示为：10000000，而用定点整数的补码表示-1，则为 11111111。
在补码表示方法中，0 的编码是惟一的，所以 n 位二进制数可以表示 2^n 个补码。

（3）反码表示法

• 定点整数的反码表示

定点整数反码的定义如下所示：

$$[X]_{反}=\begin{cases} X & 0 \leqslant X < 2^{n-1} \\ (2^n-1)+X & -2^{n-1} < X \leqslant 0 \end{cases}$$

同样，正整数的反码就是其自身，而负整数的反码可以通过对其绝对值逐位求反来求得。在反码表示法中，符号位仍然用 0 表示正号。1 表示负号。其表示的真值范围与原码相同：$-(2^{n-1}-1) \leqslant X \leqslant 2^{n-1}-1$。数值 0 用反码表示亦有+0 和-0 之分。当 n=8 时，[+0]反=00000000，[-0]反=11111111。

• 定点小数的反码表示

定点小数反码的定义如下所示：

$$[X]_{反}=\begin{cases} X & 0 \leqslant X < 1 \\ (2-2^{-n+1})+X & -1 < X \leqslant 0 \end{cases}$$

定点小数的反码求法类似定点整数的反码求法。其表示的真值范围为：
$-(1-2^{-(n-1)}) \leqslant X \leqslant (1-2^{-(n-1)})$

反码表示法中，由于 0 占有了两个不同的编码，所以 n 位二进制数可以表示 2^n-1 个反码。

（4）移码表示法

·定点整数的移码表示

$[X]=2^{n-1}+X \quad -2^{n-1} \leqslant X < 2^{n-1}$

·定点小数的移码表示

$[X]=1+X \quad -1 \leqslant X < 1$

移码的符号位与其他三种编码不同，用 1 表示正号而 0 表示负号。移码的求法其实也很简单，把其补码的符号位直接变反即可。所以在移码表示法中，0 也有惟一的表示方法：100000000（n=8）。

【例 1.10】设机器码长度为 8，求 X=（-6）10 的原码、补码、反码和移码。

解：

[X]原码=2^7-（-110）=100000000+110=10000110

[X]补码=2^8+（-110）=100000000-110=11111010

[X]反码=（2^8-1）+（-110）=11111111-110=11111001

[X]移码=2^7+（-110）=100000000-110=01111010

【例 1.11】设机器码长度为 8，求 X=（-0.3125）$_{10}$=（-0.0101）$_2$的原码、补码、反码和移码。

[X]原码= 1 -X=1.0-(-0.0101)=1.0101=10101000

[X]补码=2+X=10.0+(-0.0101)=1.1011==11011000

[X]反码=(2-2^{-7})+X=(10.0-0.0000001)+(-0.0101)=11010111

[X]移码=1+X=1.0+(-0.0101)=0.1011=01011000

实际上求某个数的原码、补码、反码和移码时不仅可以利用其定义式来求，也可以通过这 4 个编

码间的相互关系来求得。比如移码就可以通过把补码的符号位变反来求，而对于正数来讲其原码、补码和反码是完全相同的；一个负数的补码也可以很容易地从其原码推知：除符号位外，各位取反，末位再加 1 即可。

4. 浮点数据的编码方法

计算机内部浮点数据表示成如下形式：

Ms	E	M

Ms 是尾数的符号位，0 表示正号而 1 表示负号；E 是阶码，通常用移码表示；M 是定点小数形式的尾数，一般用补码或原码表示。

【例 1.12】某浮点数机器码 10 位，基数为 2，阶码用 4 位移码表示，尾数 5 位用补码表示，符号位 1 位。求 2^{10}* 0.1101 的编码。

解：

符号位 Ms=0

阶码 E=（10）$_{移码}$=1010

尾数 M=（0.1101）$_{补码}$=11010

则其编码为 0101011010

【例 1.13】某浮点数机器码为 17 位，基数为 2，阶码用 4 位移码表示，尾数用 12 位补码表示，符号位 1 位。求-2.75 的浮点的编码形式。

解：

首先把-2.75 表示成浮点形式：

（-2.75）$_{10}$=（-0.6875×2^2）$_{10}$=$(-0.1011)_2\times 2^{(10)}$

Ms=1

E=$(10)_{移码}$=1010

M=$(-0.1011)_{补码}$=010100000000

编码为 11010010100000000

同一个浮点数可以有许多种不同的编码表示（改变其阶码），所以需要对浮点数进行规格化处理，以统一其形式，同时提高数据的表示精度。

规格化处理（基数为 2）：

（1）尾数部分以纯小数形式表示，其绝对值应满足 0.5≤|M|<1。若不满足该条件，则需要修改阶码并对尾数进行移位，以使尾数满足该条件。

（2）当尾数为正时，无论其使用原码还是补码表示，M 应满足条件：0.5≤M<1。以二进制形式表示为 M=0.1xx…x(x 为 0 或 1)；

（3)用原码表示的负尾数,M 满足-<M≤-0.5,以二进制形式表示为 M=1.1xx…x(x 为 0 或 1)；

（4）以补码表示的负尾数，M 满足-≤|M|<-0.5，以二进制形式表示为 M=1.0xx…x(x 为 0 或 1)；
从上面的讨论可以看出，如果规格化的尾数用原码表示，则其最高位一定为 1，如果改用补码表示，则尾数的最高位与其符号位相反，即为 0。

1.2.3 符号数据的表示方法

计算机需要处理的数据不仅包括数值数据,而且还有大量用于表达文字信息的符号数据。这些非数值的符号数据必须通过某种编码形式用二进制位 0 和 1 来表示，才能让计算机系统对其进行处理，最常见的编码格式为 ASCII 码和 EBCDIC 码。

1.ASCII 码

ASCII 码（American Standard Code for Information Interchange）即美国标准信息交换码,该编码后来被国际标准化组织 ISO 采纳而成为一种国际通用的信息交换用标准代码，即国际 5 号码。

ASCII 码采用 7bit 位进行编码，一共有 2^7=128 种编码，从 00000000 到 11111111 可以表示 128 个不同的字符。这 128 个字符又可以分为两类：可显示/打印字符 95 个和控制字符 33 个。

所谓可显示/打印字符是指包括"0～9"十个数字符；"a"～"z"、"A"～"Z"共 52 个英文字母符号；"+"、"-"、"≠""/"等运算符号；"。"、"？"、"，"、"；"等标点符号；"#"、"%"等商用符号在内的 95 个可以通过键盘直接输入的符号。它们都能在屏幕上显示或通过打印机打印出来。

控制字符是用来实现数据通信时的传输控制、打印或显示时的格式控制以及对外部设备的操作控制等等特殊功能。共有 33 个控制字符，它们都是不可直接显示或打印（即不可见）的字符。如编码为 7DH（这里，最后一个字母 H 表示前面的 7D 用十六进制表示）的 DEL 用作删除操作，编码为 07H 的 BEL 用作响铃控制等。

ASCII 的字符编码表一共有 2^4=16 行，2^3=8 列。低 4 位编码 b3b2b1b0 用作行编码，而 b7b6b5 高 3 位用作列编码。值得注意的一点是数字"0"到"9"的编码：它们都位于 3 列（011），从 0 行（0000）排列到 9 行（1001），即"0"的 ASCII 码为（0110000）$_2$=（30）$_{16}$，"9"的 ASCII 码为（0111001）$_2$=（39）$_{16}$。把高 3 位屏蔽掉，低 4 位恰是"0"～"9"的二进制码。这个特点使得在数字符号（ASCII 码）与数字值（二进制码）之间进行转换非常方便。
计算机里的存储和传送单位通常使用 Byte(Byte=8bit),所以 7 位的 ASCII 码也用一个字节来表示。最高 1 位没有使用，通常填 0；也可以把它用作校验位或者用来扩展字符集。
2.EBCDIC 码

EBCDIC（Extended Binary Coded Decimal Interchange Code）即所谓扩展的二/十进制交换码。采用 8bit 编码来表示一个字符，共可以表示 2^8=256 个不同符号，但 EBCDIC 中并没有使用全部编码，只选用了其中一部分，剩下的保留作扩充用。EBCDIC 码常用于 IBM 大型机中。
在 EBCDIC 码制中，数字"0"～"9"的高 4 位编码都是 1111，而低 4 位编码则依次为 0000 到 1001。把高 4 位屏蔽掉，也很容易实现从 EBCDIC 码到二进制数字值的转换。

1.2.4 汉字的处理

汉字处理包括汉字的编码输入、汉字的存储及汉字的输出等环节。研究和提高汉字处理技术对于在我国推广普及计算机应用、增强我国计算机产业的竞争力、提高在国际计算机界的地位以及弘扬中华民族的文化都有级其重要的影响。

1.汉字的编码与输入

GB2312《信息交换用汉字编码字符集--基本集》收集了常用汉字 6763 个：一级汉字 3755 个，二级汉字 3008 个。

汉字的编码方法种类繁多，曾经被形容为万"码"奔腾。但主要可以分为 3 类：数字编码、拼音码和字形码。

（1）数字编码

每一个汉字都分配给一个惟一的数字代码，用以代表该汉字。国标区位码、电报码都属于该类。常用的是国标区位码（又简称国际码或区位码）。

国标区位码把 GB2312 基本集中的 6737 个汉字分为 94 个区，每个区又分 94 位，以区码和位码二维坐标形式给每个汉字进行编码。区码和位码各有 2 个十进制数字，每输入一个汉字需击键 4 次。
在 84 个分区中，1～15 区用来表示字母、数字和符号，16～87 区用以表示一级、二级汉字。其中一级汉字以汉语拼音为序排列，二级汉字以偏旁部首为序进行排列。

数字编码的特点是一字一码，无重码，编码等长，且易和内部编码进行转换。但记忆各个汉字编码的是一件极其艰巨的任务，非专业人员很难使用。

（2）拼音码

用每个汉字的汉语拼音符号作为汉字的输入编码。这种编码很容易学会使用，无需额外记忆，使用人员的负担小，所以成为最常用的一种输入方法。但是由于汉字同音字太多，重码率高，所以输入速度很难提高。

（3）字形码

以汉字的形状特点为每个汉字进行编码。最受欢迎的一种字形编码方法是五笔字型编码。其思想是依据汉字的笔划特征将基本笔划分为 5 类并分别赋以代号，见表 1.5。

表 1.5 基本笔画代号表

名称	横	竖	撇	点	折
笔画	一	丨	丿	丶	乛
代号	1	2	3	4	5

另外根据汉字的结构特征把汉字分为 4 种字型，分别赋以代号，见表 1.6

表 1.6 汉字类型的代号表

字型	左右型	上下型	包围型	单体型
代号	1	2	3	4

并把使用频率较高的常用笔划结构如"口"、"日"、"亻"、"钅"等和基本笔划一起作为汉字的构件。汉字的五笔字型编码就是依据其组成部件和结构特征进行编码。其输入能达到很高的速度。

2.汉字的存储

汉字在计算机里以内部编码（简称内码）的形式存放。通常，汉字内部编码以连续两个字节来表示。为了和英文字符的机内编码（ASCII 码）相区别，这两个字节的最高位均置"1"。这样两字节内码就可以表示 $2^{8-1}\times2^{8-1}=128\times128=16384$ 个汉字。

汉字的内码是在计算机内处理汉字信息时所采用的机内代码，与汉字的输入编码不同。与此对应，通常又把汉字的输入编码称为外码。

3.汉字的输出

汉字的输出实际上是指输出汉字的点阵字型码（又称字模码）：在显示器上显示汉字或者用打印机输出该汉字。

汉字的字型点阵码是以点阵形式描述的汉字字形代码，其点阵的密度决定了汉字的美观清晰度。普通型的点阵为 16×16，更高的有 24×24，32×32，64×64 等。汉字点阵码需要占用大量的存储空间，以普通 16×16 点阵为例：

每个汉字占用 16×16bit=32Byte

两级汉字 6763 个，共占用 6763×32=256KB

1.2.5 数据的校验方法

计算机系统在运行时，在各个部件之间经常需要进行数据交换，为了保证数据在传送过程中的正确无误，不仅需要设计高可靠性的硬件电路，同时还必须引入某种差错检查机制，对数据信息进行校验以检测是否有数据传送错误。

通常使用校验码的方法来检测数据是否出错。其基本思想是把数据可能出现的编码分为两类：合法编码和错误编码。合法编码用于传递数据，而错误编码则是不允许用来传递数据的编码。合理地设计错误编码以及编码规则，使得当数据出现某种错误后就会变成错误编码，这样就可以检测出数据传递过程是否出错。

使用校验码来查错，涉及到一个重要概念：码距。它是指在一个编码系统中任意两个合法编码之间至少有多少个二进制位不同。码距为 1 的编码是不能发现错误的。
常用的校验码有奇偶校验码、海明校验码和 CRC 校验码 3 种。

1.奇偶校验码

该编码通过通过一位校验位来使编码中 1 的个数为奇数(奇校验)或者为偶数(偶校验），从而使码距变为 2，来检测数据代码中奇数位出错的编码。因为其利用的是编码中 1 的个数的奇偶性作为依据，所以不能发现偶数位错误。以数字 0 的 ASCII 码为例，其奇校验编码为 10110000，偶校验编码为 00110000。当合法编码中奇数位发生了错误，即 0→1 或 1→0，则编码中 1 的个数的奇偶性就会改变，从而可以发现错误。该方法开销小，在很多情况下都被用作数据的绞验码。

校验位的添加方法可以有三种方式：

（1）水平奇偶校验码

对每一个数据的编码添加校验位。

（2）垂直奇偶校验码

校验位的添加不是针对单个数据，而是为一组数据的相同位添加一个校验位，形成该组数据的一个奇（偶）校验码，见表 1.7。

表 1.7 垂直偶校验码表

数据	数据编码
“0”	0110000
“1”	0110001
“2”	0110010
“3”	0110011
“4”	0110100
偶校验码	0110100

（3）水平垂直奇偶校验

先对一组数据进行垂直奇（偶）校验，所得结果（连同该垂直校验码）再添加一位水平校验位。

2.海明校验码（Hamming code）

海明码是由贝乐实验室的 Richard Hamming 设计的，也是利用奇偶性来检错和纠错。该校验码通过在数据位之间插入 k 个校验位，扩大数据编码的码距，从而不但有能力检测出错误，并能纠正错误。

若要能纠正 1 位错， k 个校验位可以有 2^k 个编码，其中一个用以表示数据无差错，而剩下 2^k-1 个编码则可用来指示哪一位数据出错了。由于 n 个数据位和 k 个校验位都有可能出错，所以 k 必须满足：$2^k-1 \geqslant n+k$

海明码的编码规则：

设 k 个检验位为 Pk Pk-1 …P1，n 个数据位为 Dn-1 Dn-2…D0，产生的海明码为 Hk+n, Hk+n-1, …H1，则有：

• Pi 在海明码的第 2^{i-1} 位置，也即有 Hj= Pi, $j=2^{i-1}$；数据位则依序从低到高占据海明码中剩下的位置。

• 海明码中的任一位都是由若干校验位来校验的。其对应关系如下：被校验的海明位的下标等于所有参与校验该位的校验位的下标之和。而校验位则由其自身来校验。

对 8 位数据位 D7 D6…D0 进行海明校验需要 4 位校验位（$2^3-1=7$，$2^4-1=15$），令其为 P4P3 P2P1。生成海明码 H12 H11 ……H1 的方法如下：

①首先确定 Di 与 Pi 在海明码中的位置：P1= H1 ， P2= H2 ， P3= H4 ， P4= H8 ，剩下则依序排列 Di，可得：

H12 H11 H10 H9 H8 H7 H6 H5 H4 H3 H2 H1

D7 D6 D5 D4 P4 D3 D2 D1 P3 D0 P2 P1

②其次确定校验关系，可得表 1.8。

表 1.8 校验关系表

海明码	下 标	校验位组	对应关系 P_4 P_3 P_2 P_1
$H_1(P_1)$	1	P_1	0001
$H_2(P_2)$	2	P_2	0010
$H_3(D_0)$	3=1+2	P_1,P_2	0011
$H_4(P_3)$	4	P_3	0100
$H_5(D_1)$	5=1+4	P_1,P_3	0101
$H_6(D_2)$	6=2+4	P_2,P_3	0110
$H_7(D_3)$	7=1+2+4	P_1,P_2,P_3	0111
$H_8(P_4)$	8=8	P_4	1000
$H_9(D_4)$	9=1+8	P_1,P_4	1001
$H_{10}(D_5)$	10=2+8	P_2,P_4	1010
$H_{11}(D_6)$	11=1+2+8	P_1,P_2,P_4	1011
$H_{12}(D_7)$	12=4+8	P_3,P_4	1100

表中最后一列对应关系表示 P4P3P2P1 中哪些位参与了校验：1 表示该检验位参与了校验，0 表示没有用到该校验位。

③确定校验位的取值：

从表中可以分别列出 P4、P3、 P2、P1 参与校验的位。

P1 校验：P1，D0，D1，D3，D4，D6

P2 校验：P2，D0，D2，D3，D5，D6

P3 校验：P3，D1，D2，D3，D7

P4 校验：P4，D4，D5，D6，D7

若使用偶校验，则

P1=D0⊕D1⊕D3⊕D4⊕D6

P2=D0⊕D2⊕D3⊕D5⊕D6

P3=D1⊕D2⊕D3⊕D7

P4=D4⊕D5⊕D6⊕D7

若使用奇校验，则 P1，P2， P3，P4 各位取偶校验值的反。

④检测错误：

对使用海明编码的数据进行差错检测很简单，只需计算：

G1= D0⊕D1⊕D3⊕D4⊕D6⊕P1

G2= D0⊕D2⊕D3⊕D5⊕D6⊕P2

G3= D1⊕D2⊕D3⊕D7⊕P3

G4= D4⊕D5⊕D6⊕D7⊕P4

若采用偶校验则G1－G4值为全 0时,数据正确(奇校验则应为全1)。当G4G3G2G1不是全0时则说明发生了差错。例如当G4G3G2G1=0101时，我们可以从表中对应关系栏找到匹配项D1，即D1这一位出错了。对其求反即可纠正该错误。

【例1.14】采用 4位校验位奇校验方式，求 10100110的海明码。

解：

D7D6D5D4D3D2D1D0=10100110，根据公式

P1 = D0⊕D1⊕D3⊕D4⊕D6= 0⊕1⊕0⊕0⊕0=1

P2 = D0⊕D2⊕D3⊕D5⊕D6= 0⊕1⊕0⊕1⊕0=0

P3 = D1⊕D2⊕D3⊕D7 = 1⊕1⊕0⊕1=1

P4 = D4⊕D5⊕D6⊕D7 = 0⊕1⊕0⊕1=0

P1 = 0; P2 = 1; P3 = 0; P4 = 1;

所求海明码为：

H12 H11 H10 H9 H8 H7 H6 H5 H4 H3 H2 H1

D7 D6 D5 D4 P4 D3 D2 D1 P3 D0 P2 P1

1 0 1 0 1 0 1 1 0 0 1 0

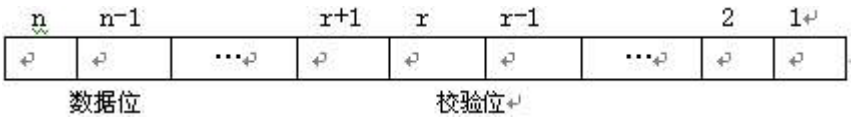
3.循环冗余校验码（cyclic redundancy check，CRC）

CRC码利用生成多项式为k个数据位产生r个校验位进行编码,其编码长度为n=k＋r，所以又称为

（n，k）码。其编码格式：

（1）编码规则

CRC 码广泛应用于数据通信领域和磁介质存储系统中。设有 k 个数据位，需要添加 r 个校验位，n=k＋r，其编码规则如下：



• 用 C（x）= Ck-1 Ck-2…C0 表示 k 个数据位，把 C（x）左移 r 位，即相当于 C（x）×2^r，给校验位空出 r 位来。

• 给定一个 r 阶的多项式 g（x），可以求出一个校验位表达式 r（x）。g（x）称为该循
环码的生成多项式。

$$\frac{C(x)\times 2^r}{g(x)} = q(x) + \frac{r(x)}{g(x)}$$

用 C（x）×2^r 去除生成多项式 g（x），商为 q（x），余数则为 r（x）。据此可得：

C（x）×2^r＝q（x）g（x）＋r（x）

在 CRC 编码过程中，四则运算采用模 2 运算，即不考虑借位和进位。所以则有：

C（x）×2^r＋r（x）＝q（x）g（x）

• C（x）×2^r＋r（x）即为所求 n 位 CRC 码，它应是生成多项式 g（x）的倍式，即可以被 g（x）整除。

• 校验数据时用数据的 n 位 CRC 编码去除 g（x），若余数为 0 则说明数据正确，否则根据余数的值即可查出差错位。

（2）模 2 运算规则

①加减法规则

0 士 0=0，0 士 1=1，1 土 0=1，1 士 1=0

没有进位和借位。

②乘法规则

利用模 2 加求部分积之和，没有进位。

③除法规则

利用模 2 减求部分余数；

没有借位；

余数最高位为 1 则商为 1，否则商为 0；

当部分余数的位数小于除数时，该余数为最后余数。

【例 1.15】求 1100000 除以 1011。

解：商为 1000，余数为 1000。

```
          1000
1011 ) 1100000
         1011
         1000
```

（3）常用的生成多项式

表 1.9 给出了几个在 CRC 编码中常用的生成多项式。

表 1.9 生成多项式表

n	k	生成多项式 $g(x)$	二进制编码
7	4	$g(x)=x^3+x+1$	1011
		$g(x)=x^3+x^2+1$	1101
7	3	$g(x)=x^4+x^3+x+1$	11011
		$g(x)=x^4+x^2+x+1$	10111
15	11	$g(x)=x^4+x+1$	10011
31	26	$g(x)=x^5+x^2+1$	100101

生成多项式应满足以下条件：

- 任何一位发生错误，C（x）×2^r＋r（x）即 CRC 码除以 g(x)的余数都不能为 0；
- 余数继续作模 2 除，应使余数循环。

若要能纠正一位错，则不同的码位错不能有相同的余数，且当给定生成多项式后，余数与出错位之间的对应关系应保持不变，与编码无关。例如 $g(x)=x^3+x+1$(1011)的查错表（一位出错）如表 1.10 所示。

表 1.10 出 错 位 表

余数	出错位	余数	出错位
000	无	011	4
001	1	110	5
010	2	111	6
100	3	101	7

【例 1.16】给定生成多项式以 g(x)=1011，用（7，4）CRC 码对 C(x)=1010 进行编码。

解：

C（x）=1010，n=7，k=4，r=3，g（x）=1011

C（x）·2^3=1010000

$$\frac{C(x)\times 2^3}{g(x)}=\frac{1010000}{1011}=1011+\frac{110}{1011}$$

则 r（x）=011，所求编码为 1010011。

【例 1.17】接上例，假设 1010011 在传送过程发生了差错，变成 C´（x)=1000011，试利用检测机制进行检查和纠正。

解：

$$\frac{C'(x)}{g(x)}=\frac{1000011}{1011}=1011+\frac{110}{1011}$$

余数为 110，查表可知第 5 位出错，对其求反则得 1010011，完成纠错。

（4）通信与网络中常用的 CRC

在数据通信与网络中，通常 k 相当大，由一千甚至数千数据位构成一帧，而后采用 CRC 码产生 r 位的校验位。它只能检测出错误，而不能纠正错误。一般取 r=16，标准的 16 位生成多项式有

CRC－16=$X^{16}+X^{15}+X^2+1$ 和 CRC-CCITT= $X^{16}+X^{15}+X^2+1$

一般情况下，r 位生成多项式产生的 CRC 码可检测出所有的双错、奇数位错和突发长度小于等于 r 的突发错以及（$1-2^{-(r-1)}$）的突发长度为 r+1 的突发错和（$1-2^{-r}$）的突发长度大于 r+1 的突发错。例如，对上述 r=16 的情况，就能检测出所有突发长度小于等于 16 的突发错以及 99.997％的突发长度为 17 的突发错和 99.998％的突发长度大于 17 的突发错。所以 CRC 码的检错能力还是很强的。这里，突发错误是指几乎是连续发生的一串错，突发长度就是指从出错的第一位到出错的最后一位的长度（但是，中间并不一定每一位都错）。

1.3 计算机算术运算和逻辑运算

1.3.1 二进制数的算术运算

1.定点数运算

（1）定点数加法/减法运算

定点数的加法和减法运算使用补码较为方便。对于定点小数的运算规则如下：

• 加法：[X+Y]补= {[X]补+[Y]补} mod2

• 减法：[X-Y]补= {[X]补+[-Y]补} mod2

当运算结果超过了定点数的表示范围时，则产生溢出（Overflow）,常用的溢出检测机制主要有以下两种：

①进位判决法

令 C_{n-1} 表示次高位（最高数值位）向最高位（符号位）的进位，C_n 则表示符号位的进位，则结果是否溢出的判决表如表 1.11 所示，即 $C_n \oplus C_{n-1}=1$ 时溢出。这里，⊕表示异或，参见后面 1.3.2 小节。

表 1.11 溢出的判决表

C_n	C_{n-1}	Overflow = $C_n \oplus C_{n-1}$
0	0	无溢出
0	1	溢 出
1	0	溢 出
1	1	无溢出

【例 1.18】X=（100）$_{10}$=（01100100）$_2$，Y=（50）\-10=（00110010）$_2$，求 X+Y。

解：

[X]补=01100100，[Y]补=00110010

[X+Y]补=[X]补+[Y]补=10010110

$C_n \oplus C_{n-1}=1$,表示结果溢出。

【例 1.19】X=（-100）$_{10}$=（-01100100）$_2$，Y=（-50）$_{10}$=（-00110010）$_2$，求 X-Y。

解：

[X]补=10011100，[-Y]补=00110010

[X-Y]补=[X]补+[-Y]补=11001110

$C_n \oplus C_{n-1}=0$,因此没有溢出。

②双符号位判决法

采用两位二进制位来表示符号位；00--正号，11--负号，根据运算结果的符号位也可以判定其是否溢出，其判决表如表 1.12 所示。

表 1.12 双符号位判决表

运算结果的符号位	Overflow	运算结果的符号位	Overflow
00	无溢出	10	溢 出
01	溢 出	11	无溢出

【例 1.20】X=（100）$_{10}$=（01100100）$_2$，Y=（50）$_{10}$=（00110010）$_2$，求 X+Y。

解：

[X]补=001100100 [Y]补=000110010

[X+Y]补=[X]补+[Y]补=010010110

其符号位为 01，说明结果溢出。

（2）定点数乘法运算

在作定点数的乘法运算时采用原码比较方便。通常使用原码一位乘法来求两个定点数的乘积。
运算规则：

• 乘积的符号位等于乘数和被乘数的符号位进行异或；

• 乘积的值等于两数绝对值之积，即乘数和被乘数的绝对值进行移位相加。
算法：（用绝对值进行运算）

令 X 表示被乘数，Y 表示乘数，P 表示部分积以存放中间结果，flag 用作判别位，count 用作计数。

• 令 P=0；count=0；flag=0；

• P 和 Y 一起右移一位（P 为高位部分，Y 为低位部分），右移时 P 的最高位补 0，P 的最低位移入 Y 的最高位，Y 的最低位移入到 flag 中；

• 如果 flag=1 同 P=P+X，否则 P 不变；

• count=count+1，如果 count 超过 Y 的位数则运算停止，否则转到上述第 2 步继续执行；

• P 和 Y 的内容就是所求乘积的绝对值，其中 P 存放积的高位，而 Y 存放积的低位。

【例 1.21】X=（10）$_{10}$=（1010）$_{2}$，Y=（-60）$_{10}$=（-0110），求 X，Y。

解：

[｜X｜]原=1010，[｜Y｜]原=0110

乘积的符号位 S=0⊕1=1，乘数和被乘数的移位相加过程见表 1.13。

表 1.13 定点数乘法运算表

执行动作	部分积 P	乘数 Y	判别位 flag	counter
初始化	0000	0110	0	0
右移	0000	0011	0	1
右移	0000	0001	1	2
P+X	1010			
右移	0101	0000	1	3
P+X	1111			
右移	0111	1000	0	4
右移	0011	1100	0	5

则所得乘积为（-00111100）$_{2}$=（-60）$_{10}$

（3）定点数除法运算

定点数的除法通常也常用原码进行。下面介绍常用的加减交替法求两个数 X 和 Y 的商。

运算规则：

• 商的符号位同定点数原码乘法的处理方法，由两数的符号位进行异或；

• 两数的绝对值部分进行相除；

算法：（求 X/Y，设 X、Y 的数值部分长度为 N）

• 令 R=X，counter=0；

• R=R-Y；若 R 为正则商 1，否则商 0；

• R 左移一位，counter=counter+1；

• 若上一次商为 1 则 R=R-Y，否则 R=R+Y；

• R 为正则商 1，否则商 0；

• 若 R=0 或者 counter≧N 则结束，否则转到上述第 3 步执行。

2.浮点数运算

在进行浮点运算之前，应先对浮点数进行规格化。

（1）浮点数的加减运算

设有浮点数 $X=M \cdot 2^i$，$Y=N \cdot 2^j$，求 X±Y，其运算过程如下：

①对阶：使两个数的阶码相同。令 K=｜i-j｜,把阶码小的数的尾数右移 K 位，其阶码加上 K。当右移尾数时，若尾数用补码表示则符号位参加移位，符号位不变；如果是原码，则符号位有参加移位，尾数的最高位补 0。

②尾数进行加、减运算。

③规格化处理：若尾数所得结果不是规格化的数，则需对其进行规格化处理。当尾数溢出时，尾数向右规格化位，阶码加 1；当尾数用补码表示，尾数的最高位与符号位相同时，应向左规格化，尾数每向左移一位，阶码减 1，直到尾数最高位与符号位相反。

④舍入操作：舍入的方法常用 0 舍 1 入，即当向右规格化时，若移掉的最高位为 1，则在尾数末位加 1；否则舍去。另一种则是"恒 1"法，即不管移走的数据为何值，尾数最末位恒置 1。

⑤溢出判断：以阶码为准。若阶码上溢，则结果溢出；若阶码下溢，则结果为 0；否则结果正确无溢出。

（2）浮点数的乘除运算

浮点数相乘，其积的阶码为两数阶码相加，积的尾数为两尾数相乘。浮点数相除，其商的阶码为两数阶码之差，商的尾数为两尾数相除。其结果都需要进行规格化处理，同时还需要判断其阶码是否溢出。

1.3.2 逻辑代数及逻辑运算

逻辑代数是 1849 年英国数学家乔治·布尔提出的代数的方式对逻辑变量进行描述和分析的数学工具，又称布尔代数。逻辑变量的取值只有"真"和"假"，通学以 1 和 0 表示。

1.基本的逻辑运算

（1）"与"（AND）运算

又称为逻辑乘运算，其运算符号通常用 AND、∩、∧或·等表示。两个变量的"与"运算的运算规则如表 1.14 所示。即当 A、B 中任一变量取 0 值时，其运算结果为 0。

表 1.14 "与"运算表

$Z=A\cdot B$	A	B
0	0	0
0	0	1
0	1	0
1	1	1

表 1.15 "或"运算表

$Z=A+B$	A	B
0	0	0
1	0	1
1	1	0
1	1	1

（2）"或"（OR）运算

又称为逻辑加运算，其运算符号为 OR、∩、∧或+等。两个变量的"或"运算规则如表 1.15 所示。即当 A、B 中任意一个取 1 值时，其运算结果为 1。

（3）"非"（NOT）运算

又称为逻辑求反运算，常用 A 表示对变量 A 进行求反，其运算规则很简单：

$\bar{1}=0$，$\bar{0}=1$。

（4）"异或"运算

常用的还有"异或"运算，又称为半加运算，其运算符号为 XOR 或⊕。异或运算是一种复合逻辑运算，可用上述基本逻辑运算表示如下：

$$A \oplus B=\bar{A}\cdot B+A\cdot \bar{B}$$

2.常用的逻辑运算公式

（1）交换律：

A+B=B+A， A·B=B·A

（2）结合律：

A+（B+C）=（A+B）+C

A·（B·C）=（A·B）·C

（3）分配律：

A·（B+C）=A·B+A·C

A+（B·C）=（A+B）·（A+C）

（4）反演律：

$\overline{A+B}=\bar{A}\cdot\bar{B}$

$\overline{A\cdot B}=\bar{A}+\bar{B}$

（5）重叠律：

A+A=A， A·A=A

（6）互补律：

$A+\bar{A}=1$， $A\cdot\bar{A}=0$

（7）0-1 律：

A·0=0， A·1=A

A+0=A， A+1=1

（8）对合律：

$\bar{\bar{A}}=A$

(9)常用公式：

$AB+A\bar{B}=A$ ， $A+AB=A$

$A+\bar{A}B=A+B$ ， $AB+\bar{A}C+BC=AB+\bar{A}C$

$\overline{A\oplus B}=\bar{A}\oplus B=A\oplus\bar{B}$

3.逻辑表达式及其化简

（1）逻辑表达式与真值表

逻辑表达式就是以逻辑运算把若干逻辑变量连接在一起表示某种逻辑关系的表达式，一个逻辑函数往往具有多种不同的表达式。我们常常用一张表来描述一个逻辑函数与其变量之间的关系--即把变量的各种取值情况都列出来，这张表称为

该逻辑函数的真值表。当变量不太多的情况下，利用真值表可以很方便地证明某些逻辑恒等式。

【例 1.22】用真值表来证明$A+\bar{A}B=A+B$ 。

证明：分别建立 $A+\bar{A}B=A+B$的真值表，见表 1.16。

表 1.16 证明 $A+AB=A+B$ 的真值表

A	B	$\bar{A}B$	$A+\bar{A}B$	$A+B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

从表中可以看出，无论 A，B 取何值，A+B 都等于 A+AB，因此公式得证。

（2）逻辑表达式的化简。

利用基本逻辑运算规律和一些常用的逻辑恒等式对逻辑表达式进行合并项、吸收项、配项、消去项等操作以达到化简该逻辑表达式的目的。

【例 1.23】化简$F=A\bar{B}+\bar{A}C+BC+CD$。

解：

$$
\begin{aligned}
F &= A\bar{B}+(\bar{A}C+BC)+CD \quad \text{（结合律）}\\
&= A\bar{B}+(\bar{A}+B)\cdot C+CD \quad \text{（分配律）}\\
&= A\bar{B}+(\bar{A}+\bar{\bar{B}})\cdot C+CD \quad \text{（对合律）}\\
&= A\bar{B}+\overline{A\cdot\bar{B}}\cdot C+CD \quad \text{（反演律）}\\
&= (A\bar{B}+A\bar{\bar{B}}\cdot C)+CD \quad \text{（结合律）}\\
&= A\bar{B}+C+CD \quad \text{（吸收律）}\\
&= A\bar{B}+(C+CD) \quad \text{（结合律）}\\
&= A\bar{B}+C
\end{aligned}
$$

【例 1.24】化简$F=AB+\overline{A\bar{C}}+B\bar{C}$

解：

$F=AB+\overline{AC}+(\bar{A}+A)\cdot B\bar{C}$ （互补律和 0-1 律）

$=AB+\overline{AC}+AB\bar{C}+\bar{A}B\bar{C}$ （分配律）

$=(AB+AB\bar{C})+(\overline{AC}+\bar{A}CB)$ （交换律和结合律）

$=AB+\overline{AC}$

1.4 计算机基本结构

1.4.1 总线结构

为了完成规定的任务，计算机的各个部件（例如 CPU、内存储器、I/O 系统）需要以某种方式互联，进行数据交换。最常见的互联结构就是总线互联结构和多总线互联结构。

总线一种是连接多个设备的信息传递通道，实际上是一组信号线。每根信号线都可以传送分别表示二进制 0 和 1 的信号。总线是一种共享的传输介质，在其上传输的信号可以被所有接入该总线的设备接受。但在同一时刻，只能有一个设备发送数据到总线上去。

为了优化设计和提高计算机系统的性能，常使用多总线互联结构。典型的计算机总线结构由内部总线和外部总线组成。内部总线用于连接 CPU 内部各个模块；而外部总线则用于连接 CPU、存储器和 I/O 系统，又称为系统总线。

1.系统总线概述

通常系统总线（图 1.3）由大约 50 到 100 条信号线组成，每一条信号线都有其特定的功能。虽然在不同的系统中，其总线的结构也各不相同，但是其信号线大都可以分为 3 类：数据线、地址线和控制信号线。

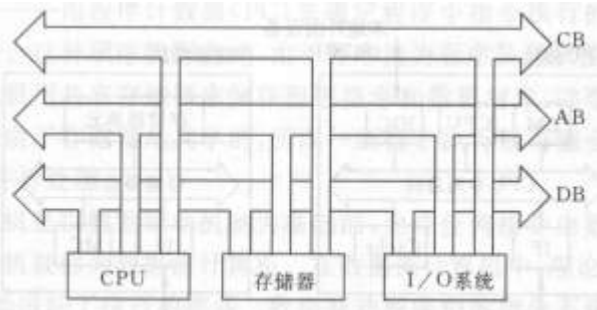


图 1.3 系统总线互联结构

数据信号线是各个模块间传递数据的通道，通常被称为数据总线（DB）。它一般由 8，16 或 32 根信号线组成。所谓数据总线的宽度就是指组成数据总线的信号线的数目，它决定了在该总线上一次可以传送的二进制位数。

地址信号线又称为地址总线（AB），用以传递地址信息，来指示数据总线上的数据的来源或去向。CPU 根据地址信息从相应的存储单元读出数据或向该存储单元写入数据。

控制信号线又称为控制总线（CB），其作用是控制数据总线和地址总线。因为多个设备共享数据总线及地址总线，所以用控制总线在各个设备间传送命令和定时信号，以协调各种总线操作。

常见的控制信号有：

• 存储器写（memory write）：使数据总线上的数据写人由地址总线上地址信号指定的存储单元；

• 存储器读（memory read）：把被寻址的存储单元中的内容放到数据总线上去；

• I / O 写（I / O write）：把数据总线上的数据输出到指定的 I/O 端口；

• I / O 读（I / O read）：从指定的 I/O 端口读取数据放到数据总线上；

• 传输响应（transfer ACK）：指示数据传输已经完成；

• 总线请求（bus request）：请求取得对总线的控制权；

• 总线允许（bus grant）：指示允许请求设备获得对总线的控制权；

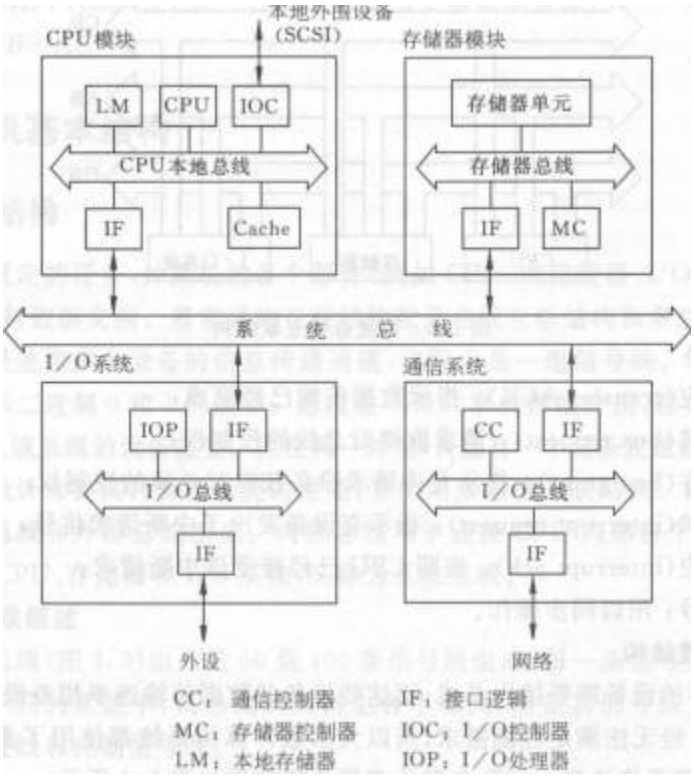
• 中断请求（interrupt request）：指示有设备发出了中断请求信号；

• 中断响应（interrupt ack）：表明 CPU 已经接受该中断请求；

• 时钟信号：用以同步操作。

2.多层总线结构

因为有许多的设备需要接入总线，而这些设备的数据传输速率相差很大，使用单一的总线进行互联已经无法满足性能需求，所以大多数计算机系统都使用了多总线体系结构进行互联，以提高系统性能。一种典型的多层总线结构如图 1.4 所示。



这种体系结构的一个优点就在于根据各个部件对数据传输率的不同要求，用不同层次的总线进行互联，以适应各自的特性与需求，不同层次间的总线相对独立，允许使用不同的信号和以不同的速率运行。对某一个层次总线结构的修改不会影响其他层次的总线结构。

3.常用的微机总线简介

（1）工业标准体系结构（industry standard architecture，ISA）
早期的 PC 机总线互联结构。ISA 采用单总线结构，数据总线宽度为 16 位，地址线宽度为 24 位，时钟为 8MHZ。

（2）扩展的工业标准体系结构（extended ISA，EISA）
提供了 32 位数据总线和 32 位地址总线，其地址空间高达 4GB（2^{32}）。向下兼容 ISA。
EISA 总线可支持 7 个 DMA 通道，15 条中断控制线。

（3）VESA 局部总线
由美国视频电子标准协会提出的一种基于多总线结构思想的互联结构。使用高速的局部总线在 CPU 和高速外设之间提供了一条高速通路。与 G、EISA 总线构成了层次结构，满足各种外设的需求。

VESA 总线控制器的缺陷在于没有提供数据缓冲器，当处理器速度高于 33MHZ 时，会引起处理延迟。

（4）外围元件互联结构（peripheral component Interconnect，PCI）

PCI 由 Intel 公司开发，并首先应用于其 Pentium 机。PCI 总线控制器在 CPU 和外设之间插人了一个复杂的管理层，以协调数据传输。PCI 提供了缓冲器，在高速时钟频率下仍能保持高性能，处理突发数据传输的能力优于 VESA 总线。目前，PCI 总线以 33MHZ 时钟运行，数据线宽度为 64bit，其吞吐量高达 264MB/s，即 2.112Gbps。

1.4.2 中央处理器

传统的冯·诺依曼（Von Neumann）计算机是基于控制流的，在用户程序中显式地说明程序的执行次序--用程序计数器（PC）来确定程序中指令执行的顺序，PC 根据程序中的指令流来定序。这种顺序执行方式，由于程序流为程序员显示控制，所以被称为控制驱动。控制流计算机用共享存储器来保存程序指令和数据对象，共享存储器中的变量可被多条指令修改。由于存储器是共享的，所以一条指令执行后可能会对其他指令产生副作用，以至于影响并行处理。

还有一种计算机是以数据驱动机制为基础的，允许任何指令由数据（操作数）可用性来驱动。这种计算机被称为数据流计算机。在数据流计算机中，理论上讲，任何指令只要操作数可用，就已经做好了执行的准备。数据驱动程序中的指令不用任何方式来排定次序。数据直接保存在指令内，而不是保存在共享存储器中。计算结果（数据令牌）直接在指令之间传送。一条指令产生的数据可以被复制成多份副本直接送给所有需要该数据的指令。

本章中所讨论的计算机硬件结构都是基于冯·诺依曼计算机模型--即控制流计算机的。

CPU 是 central processins unit 的缩写，负责完成大部分的信息处理操作。它主要由三大部分组成：寄存器组（register block）、算术逻辑单元（arithmetic logic unit，ALU）和
控制单元（control unit，CU），见图 1.5。寄存器组用以在指令执行过程中存放操作数和中间数据，由 ALU 完成指令所规定的运算和操作；CU 控制 ALU 和 Register 正确地完成某一操作。内部总线用来在寄存器之间以及寄存器和 ALU 之间提供高速的数据通路；外部总线则把 CPU 和存储器、I／O 设备连接起来完成数据交换。

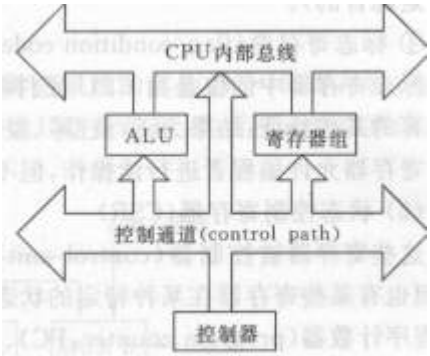


图 1.5 CPU 的基本结构模型

CPU 需要完成的功能为：

- 读取指令：从内存中读取下一条要执行的指令；
- 解释指令：对指令进行译码来确定所需执行的操作；
- 读取数据：根据指令要求从存储器或者 I / O 端口读取数据；
- 处理数据：根据指令要求对数据进行算术或逻辑运算；
- 保存数据：把执行的结果保存回存储器或者 I / O 设备。

1.寄存器组（register block）

CPU 中的寄存器通常可以分为两类：用户可见的寄存器（user-visible registers）和状态控制寄存器（control and status registers）。

（1）用户可见的寄存器

用户能够通过机器语言来访问这些寄存器。合理优化地使用这些寄存器可以减少对主存的访问次数，从而提高程序的执行速度。按其功能可以分为 4 类：

①通用寄存器（general purpose register）

通用寄存器的用途由程序员编程决定，例如可以用来存放操作数，也可以用作寻址寄存器。

②数据寄存器（data register）

数据寄存器仅能用来存放数据而不能用来对操作数的地址进行计算。

③地址寄存器（address register）

地址寄存器用来存放操作数的地址，一般而言地址寄存器又可以分为段地址寄存器（segment pointer register）、间址寄存器（index register）和堆栈寄存

器（stack pointer register）。段地址寄存器用在采用段式内存管理的计算机中，存放着某一段内存区域的基地址。间址寄存器用在间接寻址（indexed addressing）方式下。堆栈寄存器用以存放堆栈栈顶的地址，使得在使用堆栈操作（如 push，pop）时无须显式地给出操作数的地址（即其地址是隐含的）。

④标志寄存器（flag / condition codes register）

标志寄存器中的位是由 CPU 硬件根据运算的结果而设立的一些指示位，以反映该次运算的某些特征：结果为正 / 负 / 零、溢出等等，并可以用作分支跳转的依据。一般而言，标志寄存器允许编程者进行读操作，但不允许对其直接进行写操作。

（2）状态控制寄存器（CSR）

这些寄存器被控制器（control unit）用来控制 CPU 的操作。通常是程序员不可见的，但也有某些寄存器在某种特定的状态下可以由某些特权指令进行访问。常见的 CSR 有：程序计数器（program counter，PC）、指令寄存器（instruction register，IR）、存储器地址寄存器（memory address register，MAR）、存储器缓冲寄存器（memory buffer register，MBR）以及程序状态字（program status word，PSW）。

PC 中存放着下一条待取指令的地址。当程序顺序执行时，CPU 在取回一条指令后，会自动修改 PC 中的值以使其指向下一条指令。而程序中的分支或跳转指令也是通过修改 PC 中的值来达到改变流程的目的。

IR 中存放着 CPU 刚取回的指令，以供指令译码使用。CPU 和存储器之间的数据交换通过 MAR 和 MBR 来实现。MAR 中存放着待访问的内存单元的地址而 MBR 则存放从内存中读入的数据或要写回内存单元的数据。

这 4 个寄存器（PC、IR、MAR、MBR）用来实现在 CPU 和存储器（memory）之间的数据交换。另外 CPU 内部，在 ALU 与 MBR、用户可见的寄存器之间也可能存在用作输入 / 输出缓冲的寄存器。
通常 PSW 中存放着标志位（condition codes）以及状态信息。常见的标志位有：

• 符号位（Sign）：存放最近一次算术运算的符号位；

• 零标志位（Zero）；指示最近一次运算的结果是否为 0：1 表示结果为 0，0 表示结果不是 0；

• 进位位（Carry）：指示在作加法运算时是否在最高位有进位或者作减法运算最高位是否有借位；

• 相等位（Equal）：指示最近一次比较运算的结果是否相等，若相等则置 1；

• 溢出位（Overflow）：指示最近一次算术运算是否发生了溢出；

• 中断允许位（Interrupt Enable / Disable）：表示 CPU 允许或禁止中断；

• 监控位（Supervisor）：指示 CPU 是否运行在监控态，抑或是在用户态下。

在某些 CPU 中还可能有一些其他类型的 CSR 寄存器，例如中断向量寄存器（interrupt vector register）；指向其他状态信息、内存区域（如进程控制块，process contrd block）的指针寄存器；用在支持虚拟存储器的系统中的页面寄存器（page table pointer register）以及用于控制 I / O 操作的寄存器等。

2.运算器（ALU）

ALU 负责对数据进行算术和逻辑运算。计算机系统的其他部分，即寄存器、存储器以及 I/O 设备把待处理的数据输入到 ALU，由 ALU 进行处理，然后把运算结果输出，同时设置标志位。如图 1.6 所示。

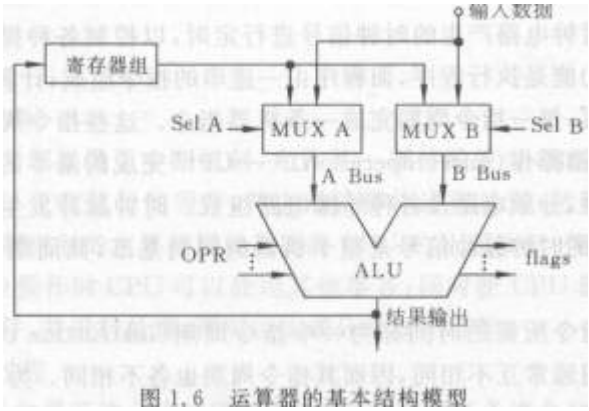


图 1.6 运算器的基本结构模型

ALU 处理的数据位数与机器位数有关，对于 IBM PC / XT、AT 而言 ALU 是 16 位；而 386、486 机器 ALU 则为 32 位。ALU 一般具有下面几种功能："A＋B"、"A-B"、"B-A"、"A OR B"、"A AND B"、"A XOR B"等功能，还可以完成左移和右移功能。

3.控制器（CU）

控制器（见图 1.7）负责控制整个计算机系统的运行，读取指令寄存器、状态控制寄存器以及从外部来的控制信号（例如中断信号），发布外控制信号控制 CPU 与存储器、I/O 设备进行数据交换，发布内控制信号控制寄存器间的数据交换，控制 ALU 完成指定的运算功能，并且管理其他的 CPU 内部操作。

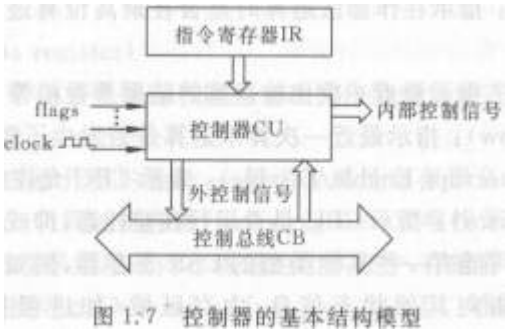


图 1.7 控制器的基本结构模型

（1）控制器的基本功能

控制器的基本功能就是时序（sequencing）控制和执行（execution）控制。根据当前运行的程序，控制器使 CPU 按一定的时序关系执行一系列的微操作（micro-operation），从而完成程序规定的动作。控制器的输人信号有：时钟信号、指令寄存器、标志位、控制总线上的控制信号。控制器的输出信号包括：CPU 内的控制信号、发往控制总线的控制信号。

①时钟信号

控制器根据由时钟电路产生的时钟信号进行定时，以控制各种操作按指定的时序进行。计算机的基本功能是执行程序，而程序由一连串的指令组成；计算机的执行过程由一连串的指令周期组成，每一指令周期完成一条机器指令。这些指令周期又可进一步细分为更小的单元，直到微操作（micro-operation）--CPU 完成的基本的原子操作。时钟电路由时钟脉冲发生器、分频电路及各种外围电路组成。时钟脉冲发生器的晶振频率称为机器的主频，它产生的时钟脉冲信号是整个机器的时间基准，其周期 T 称为该计算机的时钟周期。

执行一条机器指令所需的时间称为一个指令周期（instruction cycle），因为 CPU 机器指令的执行的时间通常互不相同，因而其指令周期也各不相同。为了便于控制和管理，把指令周期进行分解，细化为各种原子操作（微操作，μop），每一 μop 都与 CPU 的寄存器相关。完成一个微操作的时间就称为 CPU 周期（机器周期）。例如一条指令周期通常可分为取指子周期、执行子周期、保存结果子周期，而每一个子周期又由若干原子周期--CPU 周期组成。

【例 1.25】假设某微处理器的主频为 20MHZ，两个时钟周期组成一个机器周期，平均三个机器周期可完成一条指令，则其时钟周期为_____ns，平均运算速度为_____ MIPS；若机器的主频为 10MHZ，则其时钟周期为_____ns，平均运算速度为_____ MIPS。

解：设机器的主频为 Osc（HZ），m 个时钟周期组成一个机器周期，且 I 个机器周期完成一条指令，则

机器的时钟周期 T=1 / Osc（s）

平均运算速度 V=1 / mIT=Osc / mI（IPS）

（a）Osc=20MHZ，m=2，I=3

T=1 / 20MHZ=50（ns）

V=Osc / mI≈3.33（MIpS）

（b）Osc=10MHZ，m=2，I=3

T=1 / 10MHZ=100（ns）

V=Osc / mI≈1.66（MIPS）

②指令寄存器（IR）

控制器需要完成取指令、分析指令和执行指令的操作。控制器根据程序计数器PC中的内容（指令地址）从存储器中取出该指令；然后对指令进行译码以确定所需执行的 μop；最后控制器根据分析的结果发出一系列的控制信号控制各相关部件（运算器、存储器、I/O设备）完成规定的操作；同时产生下一条指令的地址。

MAR←（PC）

MBR←Memory

IR←（MBR）

PC←（PC）＋1

③中断控制逻辑

计算机系统通常都提供了中断机制，允许某一事件的发生（如由I/O设备提出服务请求）可以中止CPU正在执行的程序，转去对该事件进行处理，然后再返回原程序被中止处继续执行。中断机制的主要作用就是提高CPU的处理效率；使CPU与I/O设备并行工作，当进行I/O操作时 CPU可以处理其他事务；同时使CPU能够及时地响应各种预先不知的异常事件，并进行处理；还可以实现分时操作等等。

（a）中断处理过程

如图1.8所示，如果系统允许响应中断请求，那么在每条指令执行完成后，控制器都要检查是否有中断发生：如果没有中断请求信号，控制器则转去取下一条指令；如果有中断事件发生，则暂停执行当前程序并保护其现场状态信息（如下一条指令的地址、相关寄存器的内容等等），然后转去执行中断服务程序。当完成中断服务后，再恢复以前保存的现场信息，从原程序被中止处（断点）继续执行。

通常把CPU中断处理过程（见图1.9）分为两个阶段：中断响应过程和中断服务过程。

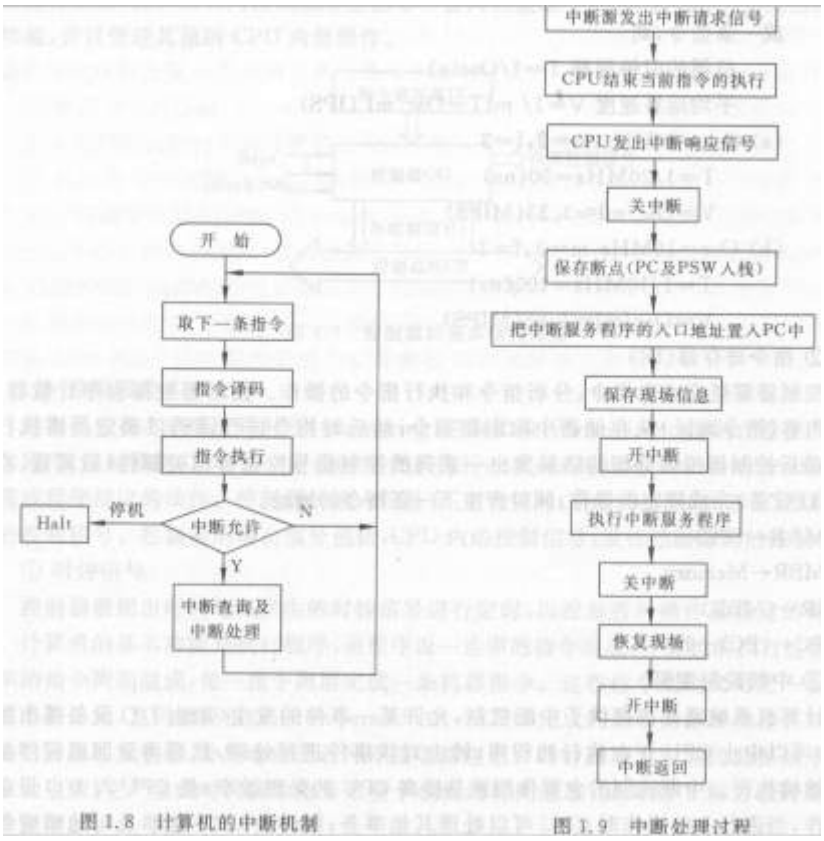


图 1.8 计算机的中断机制 图 1.9 中断处理过程

中断响应过程

CPU 接受中断请求后，就暂停执行当前程序，进入中断响应过程：

• 关闭中断：为了使 CPU 在中断服务后能够返回原程序断点处继续执行，在中断响应期间，保存断点信息之前，需要关闭中断允许，禁止 CPU 响应新的中断请求。

• 保存断点信息：当 CPU 进人中断响应过程，关闭中断允许后，控制器要保护当前 正在执行的程序的断点信息：把程序的断点即 PC 里的内容，以及 PSW 里的内容等推入系统堆栈。中断返回以后，再从堆栈中恢复现场信息，继续执行主程序。

• 把中断服务程序的入口地址置人 PC 中：控制器保存完断点信息后，接着就把相应的中断服务程序人口地址置入程序计数器中去，CPU 马上要执行的下一条指令就是中断服务程序的第一条指令。

中断服务过程

当 PC 中置入中断服务程序的人口地址之后，接着保留现场信息，CPU 开放中断允许响应新的中断请求，然后就转入中断服务程序，对中断事件进行处理。中断服务完成之后，CPU 再次禁止中断，以恢复现场和断点信息，使之返回到原主程序断点处继续执行。

（b）中断的种类

按中断源的位置分

根据发出中断请求的对象的位置可以把中断分为：

• 内部中断（陷阱）：发生在 CPU 内部（机器内部状态）或由软件指令触发。如：除法错中断、溢出中断、单步中断、软件中断、电源故障中断、时钟中断、控制器中断（非法指令、特权指令）等。

• 外部中断：由外部设备产生的中断请求，例如键盘中断、打印机中断、I／O 设备中断等。

按中断源的类型分

• 硬件中断：由硬件设备提出中断请求。

• 软件中断：由中断指令引起的中断。

按中断源的屏蔽特性分

• 可屏蔽中断：受中断屏蔽寄存器控制，可以由软件设置是否响应该中断。

• 不可屏蔽中断：不受中断屏蔽寄存器的影响，如电源故障中断。

（c）多重中断

当有多个中断源都向 CPU 发出中断请求时，CPU 有两种处理策略（见图 1.10）。

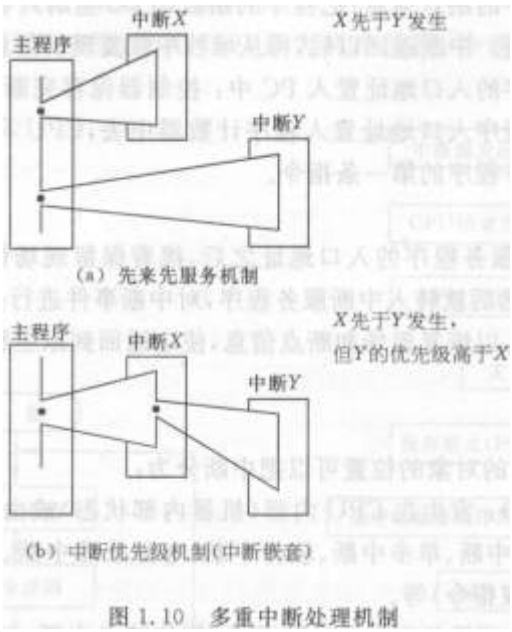


图 1.10 多重中断处理机制

先来先服务（中断排队）

当 CPU 接受一个中断请求后，立即屏蔽掉其他中断，直到该中断返回后，才响应下一个中断请求。这种机制简便且易实现，但没有考虑到不同的中断有不同的优

先级别，很难适应某些对实时性要求很严（time-critical）的任务的需求，即高优先级的中断事件需要立即处理。

基于优先权（中断嵌套）

允许优先级高的中断请求可以中止低先级低的中断服务程序，先服务高级别的中断请求，再返回原中断程序继续执行。各中断的优先级可以由硬件决定也可以通过编程设定。

④总线控制逻辑

计算机各个部件通过总线（BUS）相互连接起来，传递信息协调工作，共同完成操作。CPU 内部各器件之间互连的总线称为内部总线，CPU 与其他部件之间互连的总线称为外部总线（又称系统总线）。

由于总线是共享的传输通道，任意一时刻只能由一个器件负责发送数据，所以需要由总线控制器来协调各部件申请使用总线。当发生多个设备争用总线时，仲裁电路会依据某种策略（如优先级）来决定使用总线的先后顺序。常用的仲裁方式主要有中央仲裁和分布式仲裁两种。

（2）控制器的实现（硬布线逻辑 / 微程序控制）

控制器的作用就是根据输入的时钟信号、指令信息以及各种状态信号产生相应的控制信号去控制各个部件完成特定的操作。控制器的逻辑设计中包括时序逻辑（控制微操作的时序关系）、译码逻辑（对指令操作码进行译码分析）、执行逻辑（按一定的顺序执行微操作）和决策逻辑（根据 ALU 等部件的标志位来决定下一步的动作）。控制器的实现方法主要有两种：硬布线逻辑（hardwired implementation）和微程序控制（micro-programmed implementation）。

①硬布线逻辑

在这种机制下，控制器基本上由组合电路实现，因此这种方法又称之为组合逻辑实现（图 1.11）。

（a）指令译码

控制器分析每条待执行的指令，对不同的操作码产生不同的控制信号。为了简化设计，可以由一个译码器来完成操作码的识别功能。一个 n 位输入的译码器可以识别 2n 条不同的指令操作码，为每一条操作码产生不同的输出信号。

（b）定时（一个指令所产生的输出控制信号是有时序关系的）控制器利用输入的时钟信号作为产生各种控制信号的定时信号。因为控制器在一个指令周期内，需要在不同的时刻发出不同的控制信号，所以采用一个定时电路根据时钟信号依次产生定时信号如 T1，T2，…，Tn。控制器就在 T1，T2，…，Tn 的作用下，顺

序产生各种控制信号。在指令周期的末尾，控制器要复位定时电路，以便在下一个指令周期开始时，定时电路又可以重新产生定时信号 T1，T2，…，Tn。

（c）控制逻辑

我们已经讨论了硬布线逻辑控制器的定时逻辑和指令译码逻辑，接下来讨论其内部控制逻辑。控制逻辑的功能就是根据输入信号产生相应的输出控制信号，所以可以用一组布尔方程来描述控制器以及 CPU 的行为。

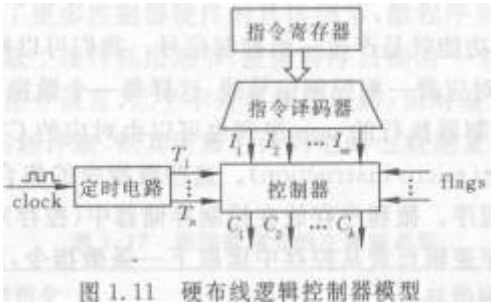


图 1.11 硬布线逻辑控制器模型

随着计算机技术的发展，CPU 的结构越来越复杂，用来描述控制器行为的布尔方程的数目随之急剧膨胀。用组合电路来实现控制器，满足所有的布尔方程，也越来越艰难了。用组合逻辑电路来设计控制器不仅复杂而且难以测试，并且也很不灵活，例如为了增加一条新的机器指令，就需要对组合逻辑电路进行修改，通常这种修改是很难实现的。

一种相比之下非常简单的方案--微程序控制逐渐被用于控制器的设计和实现过程中。

②微程序控制（micro－programmed control）

微程序控制的思想最早是由 M.V.Wilkes 在 20 世纪 50 年代初期提出的。因为受当时存储器技术的限制，并未真正得到应用。但随着半导体存储器技术的发展，今天微程序控制技术已经在许多领域里得到广泛应用，其中之一就是用来实现处理器中的控制单元（图 1.12）。现代的 CISC 处理器大多采用微程序控制方式；而 RISC 处理器由于简化了指令系统，其结构也随之而简化，所以也可以使用硬布线逻辑。

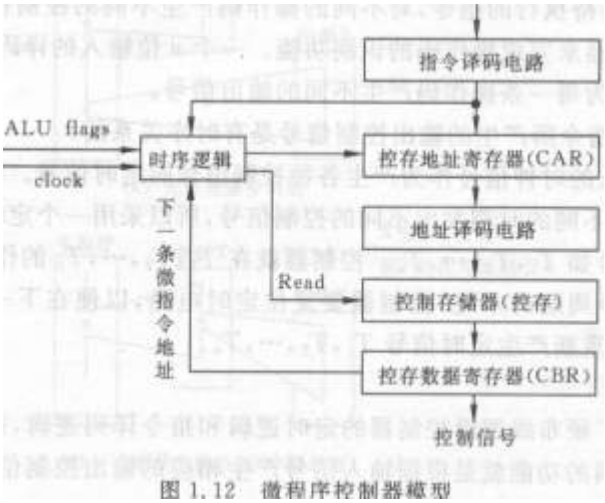


图 1.12 微程序控制器模型

（a）基本原理

对于控制器而言，其功能就是产生一组控制信号。我们可以构造一个控制字（control word，CW），其每一位都对应着一根控制信号线，这样每一个微操作都可以由控制字取不同的值来表示。同样，控制器执行的 μop 序列也可以由对应的 CW 序列来表示。每一个 CW 都代表着一条微指令（micro-instruction）。这些微指令的集合就叫做程序，每一条机器指令都对应着一段微程序。微程序存放在控制存储器中（控存）。

微程序控制器的时序逻辑负责从控存中读取下一条微指令，而执行机构则根据当前的微指令产生所需的控制信号。

（b）流程

- 控存地址寄存器（CAR）中存放着下一条微指令的地址；
- 时序逻辑发出对控存的读命令，读取下一条微指令；
- 控制器把由 CAR 指定的控制字读入控存数据寄存器（CBR）；
- 根据 CBR 中的内容产生控制信号；
- 根据 CBR 中的内容以及 ALU 的标志位等状态信息，产生下一条微指令的地址，并将其置入 CAR。

（c）微指令格式

微指令由两部分组成：控制码和地址码。控制码部分用于产生控制信号，而地址码部分则参与形成下一条微指令地址。在 Wilkes 最初的设计里，微指令里每一 bit 都直接产生一种控制信号或直接形成下一条指令地址的一位。但是由于整机所需的控制信号可能多达上百个，这样会使微指令的长度变得难以实现。对控制

信号进行编码，可以有效地减少微指令的长度。所以微指令主要可以分为两类：水平型和垂直型，见图 1.13。

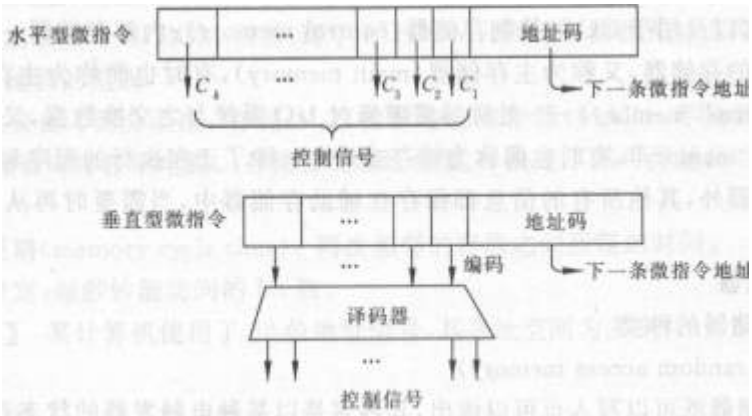


图 1.13 微指令格式

水平型微指令体现了更多控制器硬件的具体细节，微程序员可以控制所有的控制信号。而垂直型微指令屏蔽了硬件低层细节，使微程序员面对一个相对高层的控制器硬件进行编程。对水平型微指令而言，几乎不需要译码电路；而对垂直型微指令来说，指令屏蔽的硬件细节越多，编码越详细，则其所需的译码电路也就越复杂。表 1.17 列出了两种微指令的主要特点。

表 1.17 两种微指令的主要特点表

水平型微指令	垂直型微指令
指令长度较长	指令长度较短
体现了更多的硬件细节	屏蔽了一些低层硬件细节
难以编程	容易编程
并行度高	并行度较低
控制逻辑简单	控制逻辑复杂
执行速度快	执行速度较慢
对性能优化	对编程优化

微程序存放于ROM或EPROM存储器中，这种软硬一体的形式称为固件(firmware)，它的特性介于软件（software）与硬件（hardware）之间。若控存允许改写，则微程序控制被称为是动态的，即允许对指令系统进行修改。

1.4.3 存储器系统

计算机的存储器系统由分布在计算机各个不同部件的多种存储设备组成；位于CPU 内部的寄存器，以及用于 CU 的控制存储器（control memory）；内部存储器--可以被处理器直接存取的存储器，又称为主存储器（main memory），有时也简称为主存或内存；外部存储器（external memory）--处理器需要通过 I/O 系统与之交换数据，又称为辅助存储器（auxiliary memory），有时也简称为辅

存或外存。除了正在执行的程序和处理的数据存放在主存储器外，其他所有的信息都保存在辅助存储器中，当需要时再从辅存中调入主存。

1.主存储器

（1）主存储器的种类

①RAM（random access memory）

RAM 存储器既可以写入也可以读出，其内容是以某种电触发器的状态存储的，断电后信息无法保存（volatile），因此只能用于暂存数据。RAM 又可分为 DRAM 和 SRAM 两种：DRAM（dynamic RAM）信息会随时间逐渐消失，因此需要定时对其进行刷新（refresh）维持信息不丢失。

SRAM（static RAM）在不断电的情况下信息能够一直保持而不会丢失。DRAM 的密度大于 SRAM 且更加便宜，但 SRAM 速度快，电路简单（勿需刷新电路），然而容量小，价格高。

②ROM（read only memory）

只读存储器，信息已固化在存储器中。ROM 出厂时其内容由厂家用掩膜技术（mask）写好，只可读出，但无法改写。一般用于存放系统程序 BIOS 和用于微程序控制。

③PROM（programmable ROM）

可编程 ROM，只能进行一次写入操作（与 ROM 相同），可以在出厂后，由用户使用特殊电子设备进行写入。

④EPROM（erasable PROM）

可擦除的 PROM，其中的内容既可以读出，也可以写入。但是在一次写操作之前必须用紫外线照射 15～20 分钟以擦去所有信息，然后再用特殊的电子设备（EPROM 编程器）写入。可以写多次。

⑤ E^2PROM（electrically EPROM）

电可擦除 PROM，与 EPROM 相似，可以读出也可写入，而且在写操作之前，不需要把以前内容先擦去，能够直接对寻址的字节或块进行修改。只不过写操作所需的时间远远大于读操作所需时间（每字节需几百毫秒），其集成度也较低。

⑥闪速存储器 FM（flash memory）

其特性介于 EPROM 与 E^2PROM 之间。类似于 E2PROM，闪速存储器也可使用电信号进行删除操作。整块 Flash Memory 可以在数秒内删除，速度远快于 EPROM（15 一 20 分钟）。可以选择删除某一块而非整块芯片的内容，但是还不能进行字节

级别的删除操作。FM 的集成度与 EPROM 相当，高于 E2PROM。闪速存储器有时也简称为间存。

（2）主存储器的组成

存储器主要由 4 部分组成：存储体、存储器地址寄存器（MAR）、存储器数据缓冲器（MDR）和存储器时序控制电路。存储器的地址位数 N 决定了其可寻址的存储单元的个数 $M= 2^N$
存储器周期（memory cycle time）：两次相邻的存取之间所需的时间。

存储器带宽：每秒钟能访问的 bit 数。

【例 1.26】某计算机使用了 32 位地址信号，其寻址空间为多少？

解：

N=32
$M=2^N=2^{32}=4$ G Byte

【例 1.27】如果存储器周期是 500ns，而每个周期可访问 4Byte，其带宽为多少？

解：

T＝500 ns，
带宽（Bandwidth）=（1/（$500*10^{-9}$））*4=$8*10^6$（Byte/s）
=64M(bit/s)

2.辅助存储器

辅助存储器用于存放当前不需要立即使用的信息。一旦需要，再和主机成批交换数据，是主存储器的后备，因此称之为辅助存储器；它又是主机的外部设备，又称之为"外存储器"。辅助存储器的最大特点是存储器容量大、可靠性高、价格低。其主要可分为磁表面存储器和光存储器。

（1）磁表面存储器

磁表面存储器通过磁头来进行读写。磁头线圈中通入一定方向和大小的脉冲电流，磁化磁头的导磁体，从而建立起一定方向和强度的磁场。当磁介质从磁头下经过时，从磁头间隙处将其磁化。由于剩磁效应的存在，当磁场消失后，磁介质表面仍存在剩磁。线圈中电流方向不同，磁介质上被磁化的方向也不同--用以代表"1"和"0"。随着电流的变化以及磁介质磁头间的相对运动，就可以把二进制信息序列转化为介质表面的磁化单元序列。
磁表面存储器目前主要有磁带存储器和磁盘存储器两大类。

①磁带存储器

磁带存储设备是一种顺序存取的设备，存取时间较长，但存储容量大，便于携带，价格便宜，所以也是一种主要的辅助存储器。磁带的内容由磁带机进行读写（最便宜也最慢）。

（a）工作原理（读写）

按磁带机的读写方式主要可以分为两种：启停式和数据流式。

启停式磁带机

磁带按宽带可以分为 1／4 英寸、1／2 英寸和 1 英寸三种。磁带上的信息以文件块的形式存放。整盘磁带的开始有一卷标标明，然后有一初始空白块，用以适应磁带从静止到稳定带速所需的时间。文件记录以文件头标志和文件尾标志标识，一个文件由若干数据块组成，每一数据块又由若干记录组成。数据块之间以空白块进行分隔（一个数据块所包括的记录条数叫块因子），文件之间也存在一段空隙 G。所有的文件都顺序地排列在磁带上，一个文件的长度不仅包括记录信息，也包括块间间隔。

磁带机每一次读写信息的位数与磁带表面并行记录信息的磁道数有关：如 7 道、9 道和 16 道，则分别有 7，9，16 个磁头并列，一次可以读写 7 位、9 位或 16 位。

数据流磁带机

其记录格式是串行逐道记录信息，每次读写 1 位信息，数据连续地写在磁带上，数据块之间以空隙分隔。磁带机不能在块间启停。读写顺序如下：（4 个磁道）先从 2 道的首端（BOT）开始，到其末端 EOT；然后第 1 道反向记录从 EOT 到 BOT，而 2 道又正向从 BOT 到 EOT，3 道又反向。数据流磁带机结构简单，价格低，数据传输速率快。

（b）性能参数（启停式）

• 数据传输速率（byte／s）=磁带记录密度（byte／mm）×带速（mm／s）

• 数据块的长度=B1（记录数据所需长度）＋B2（块间间隔）
B1=（字节数／记录）×块因子／记录密度

• 读 N 条记录所需时间 T=S（启停时间）＋R（有效时间）＋D（间隔时间）
R=（N×字节数／记录）／传输速率
D=块间隔总长／带速=｛（N÷块化因子）×（块间间隔）｝／带速。

②磁盘存储器

（a）磁盘存储器概述

磁盘上的数据都存放于磁道上。磁道就是磁盘上一组同心圆，其宽度与磁头的宽度相同。为了减少干扰，磁道与磁道之间要保持一定的间隔（inter-track gap），沿磁盘半径方向，单位长度内磁道的数目称之为道密度 TPI（道 / 英寸）。最外层的磁道为 0 道。

位密度：沿磁道方向，单位长度内存储二进制信息的个数叫位密度。为了简化电路设计，每个磁道存储的 bit 数都是相同的，所以磁盘的位密度也随着磁道从外向内而增加。

磁盘的数据传输是以块为单位的，所以磁盘上的数据也以块的形式进行存放。这些块就称为扇区（sector），每个磁道通常包括 10～100 扇区。同样为了避免干扰，扇区之间也相互留有空隙（inter-record gap）。

柱面：n 个磁盘组成的磁盘组，所有盘面上相同位置的磁道组称为一个柱面（每个柱面有 n 个磁道）；若每个磁盘有 m 个磁道，则该磁盘组共有 m 个柱面。

为了让磁头读写定位机构能够确定磁盘上某一磁道的具体位置，因此在新磁盘使用之前，必须对其进行格式化，以写入控制数据供磁盘驱动器使用，该数据对用户不可见。从没有格式化过的盘叫做白盘，上面只是记录了有多少磁道，磁道密度等全球统一标准，还不能直接使用。

通常，在计算机系统中使用的磁盘存储器分为硬盘和软盘两种。硬盘的存储介质是铝合金等"硬"材料，并在其表面覆以磁性材料；而软盘的介质是以塑料等软材料制成。软盘成本低、重量轻、易携带易保存，给计算机信息的转储、拷贝提供了极大的方便；但速度慢、易损坏、存储容量小（一般有 1.2MB 和 1.44MB 两种）。硬盘容量大（可达十几 GB）、读写速度快、可靠性高，但一般固定在计算机内部，不可随意拆卸和移动。

（b）性能参数

• 存储容量：磁盘存储器格式化后能够存储有用信息的总容量为：

capacity=n×t×s×b

n：存放数据的总盘面数；

t：每面磁道数；

s：每道的扇区数；

b：每个扇区存储的字节数。

• 存取时间（access time）

• 寻道时间（seek time，查找时间）为磁头移动到目标磁道（或柱面）所需的时间。

·等待时间(rotational latency)为待读写的扇区旋转到磁头下方所用的时间。一般选用磁道旋转一周所用时间的一半作为平均等待时间。

寻道时间由磁盘机的性能决定，目前主流硬盘典型的 AST（average seek time）一般在 10ms 左右，而转速则有 2400rpm，5400rpm，7200rpm 等等。软盘转速较慢，一般只有 360rpm（磁头与盘面接触性读写）。

·数据传输速率 R：磁头找到地址后，单位时间写人或读出的字节数。

R=B/T

B：一个磁道上记录的字节数；

T：每转一周所需时间。

【例 1.28】某磁盘格式化为 24 扇区和 20 磁道。该盘可顺时针和逆时针旋转。各种操作时间如下：盘旋转一圈时间为 360ms，磁头从中心通过 20 磁道移动到边缘的时间为 40ms。求该盘的平均等待时间和平均寻道时间？

解：

因为该盘可顺时针和逆时针旋转，所以最短的等待时间为 oms，而最长的等待时间等于磁道旋转一周所用时间的一半，即 180ms；所以平均等待时间为 90ms。

对寻道时间而言，最佳情况是磁头位于中间磁道处，此时最短寻道时间为 0ms，最长寻道时间为 20ms，其平均值为 10ms；最坏情况是磁头位于最边缘或者最中心处，此时最短寻道时间为 oms，最长寻道时间为 40ms，其平均值为 20ms。所以该盘的平均寻道时间为最佳和最坏两种情况的平均值 15ms。

③RAID 存储器

因为辅助存储器性能改进的速度远远跟不上处理机以及主存储器性能的改进，所以存储系统也就成为改善整个计算机系统性能的关键。并行系统结构提供了一种采用低成本器件技术获得高性能的方法。

1988 年美国加州大学伯克利分校 D.A.Patterson 教授提出了廉价磁盘冗余阵列（redundant array of inexpensive disks，RAID）技术，旨在缩小日益扩大的 CPU 速度和磁盘存储器速度之间的差距。其策略是用多个较小的磁盘驱动器替换单一的大容量磁盘驱动器，同时合理地在多个磁盘上分布存放数据以支持同时从多个磁盘进行读写，改善了系统的 I/O 性能。小容量驱动器阵列与大容量驱动器相比，具有成本低、性能好的优势；低代价的编码容错方案在保持阵列的速度与容量优势的同时保证了极高的可靠性。同时也能较容易地扩展容量。

④光存储器

光盘存储器是利用激光束在记录介质表面存储信息，根据激光束的反射光来读出信息。自从 1983 年 CD 数字音频系统大获商业成功以来，极大地推动了发展低成本的光存储器技术作为计算机系统辅助存储器。一系列的光盘（optical-disk）产品纷纷推出：CD，CryROM，WORM 以及 EOD。

（a）CD（compact disk）

CD 是指用于存放数字音频信息。标准系统使用 12cm 的光盘，可以存储超过 60 分钟不间断的音频信号。

（b）CD-ROM（compact disk read-only memory）

CD-ROM 是由在塑胶圆盘的表面覆以一层高反光物质--通常是铝（aluminum）制成的。数字化编码信息是以其反光表面一系列极其微小的凹坑来记录的。

CD-ROM 的读出是通过用低功率的激光束来照射盘面，有凹坑和无凹坑处的反射光强度不相同，光电检测电路把这些再转换为数字信息即可读出信息。

CD-ROM 非常适用于把大批量数据分发给大量的用户。与传统磁盘存储器相比，具有以下优点：

- 具有更大的容量、可靠性高；
- 光盘的复制更简易；
- 可更换，便于携带，大多硬磁盘是可移动的

其缺点：

- 只读；
- 存取时间还比较长。

常用的 CD-ROM 驱动器已经从单倍速（150KB/s）光驱发展到了 24 倍速（360KB 乃）光驱。

（c）WORM（write－once read-many）

WORM 盘允许用户使用一种比 CD-ROM 较昂贵的驱动器对其写入数据，一旦写过之后，即不能再进行写操作但可以随意读。

WORM 的制作过程如下：先用高功率的激光束照射光盘，使之在表面产生一系列的泡（blister），即可提供给用户使用。

WORM 的写入过程：当已经预格式化的盘放入到 WORM 驱动器后，使用功率较低的激光束照射光盘，其强度刚好使气泡破裂（burst blisters）。通过使气泡破裂

或不破裂就可以记录信息。WORM 的读出也很简单，用激光束照射盘面，很容易识别出气泡破裂处，使用比较简单的电路即可完成读操作。

（d）EOD（erasable optical disk）

EOD 盘可以反复地进行读写，如同普通的磁盘。最常用的是磁光盘。利用磁光效应来进行读写。其写入时，用激光束来加热介质，从而改变（磁光材料介质）其局部磁场的方向来记录信息。由于该极化过程（polarization process）并不会引起光盘物理特性的改变，所以该过程可以反复进行。其读出也是利用磁光效应来进行的。

1.4.4 I/O系统

I/O 系统是计算机与外界进行数据交换的通道，但它不仅仅只是把外部设备和系统总线简单地连接起来，而且还具有某些"智能"：即 I/O 系统分担了部分数据通信的控制任务。因为随着计算机技术的发展，外设的种类日益增加，其控制方式也各不相同，很难做到由 CPU 来统一控制这些设备；外设的数据传输率也千差万别，把高速的系统总线和低速的外设直接相连会降低系统的工作效率；同时外设的数据格式也不一定会和计算机相同，无法直接进行数据交换。所以需要一个 I/O 系统在 CPU、存储器和各种外部设备之间负责协调和控制数据的输入/输出。

1.常见的外部设备

（1）输入设备

①键盘

在今天的计算机系统中，键盘早已取代了早期的纸带、卡片输入机而成为最常用的标准输入设备。主要作为字符和数字的输入设备，汉字也可以通过特定的编码系统输入。目前常用的标准键盘为 101 键。

②鼠标器，跟踪球（轨迹球），操纵杆

这三种设备的工作原理类似，主要应用于图形界面的应用环境中，通过移动光标的方式完成选择执行命令的功能。

• 鼠标器

一种手持式的坐标定位部件，按其工作原理主要可以分为机械式、光电式和混合式 3 种。

机械式鼠标器是靠其底座的圆球随鼠标在平面上运动时而滚动，从而带动与其接触的传感器（电位器）测出其上下左右的相对位移量，以此来控制屏幕上鼠标的移动。

光电式鼠标器的底座上安装有发光管，当鼠标器在专用的反光板上移动时，根据反光来确定相对位移量。

混合式鼠标底座装有圆球，圆球移动带动其内部的光栅轮，每个光栅轮装有两个光电管，可以测出栅轮的运动从而确定鼠标的相对位移量。

根据鼠标器上的功能键一般又可以把鼠标分为 2 键和 3 键。2 键鼠标器其左键一般用作确定键（相当于回车键），而右键一般用作取消键（相当于 Esc 键）。当然其功能还是主要由应用程序来决定。

• 轨迹球

其原理类似于鼠标器，只不过其一般固定在键盘边上，通过用手拨动圆球来移动光标。

• 操纵杆

其原理也类似于鼠标器，固定在桌面上，通过移动操纵杆来控制光标的移动。

③光笔输入

外形如同一支笔，顶端的透镜系统可以把进入的光聚成一个光点。将其顶端对准屏幕，按下笔上的开关即可获取屏幕上的坐标，在屏幕上移动光笔即可得到其运动轨迹。

④图像输入设备

摄像机是最常见的图像输入设备。把摄入的图像经数字化后存入到磁盘等存储器中。图文扫描仪可以把平面图像扫描后输入计算机。

⑤语音输入设备

能够把语音信号输入到计算机系统中，并对其进行识别以领会其含意。目前仍处于研究与试制阶段，通过学习过程识别有限集合的命令的技术已趋成熟。

⑥光学字符识别（OCR）

通过扫描输入字符的图像，并识别出其代表的字符。由于印刷体字相对较规范因而识别率较高，相形之下手写体的识别就较困难了。

⑦条形码输入

条形码已经在各行各业得到了广泛应用，其工作原理是用粗细不同的明暗条纹来表示数码等各种信号，利用光电效应，把这些条纹分别转换成不同的电信号，经过译码即可知道其对应的符号。利用它，可以很方便地把条形码编码信息，输入到计算机中。

⑧中文手写输入系统

该设备在软件的配合下，可边书写边识别边输入汉字。即使在单个汉字连笔书写的情况下，目前已可达到实用阶段的识别率。

（2）输出设备

常见的输出设备主要有显示设备、打印设备、绘图仪设备以及音频信号的输出设备。

①显示设备

显示设备按显示器件分类有阴极射线管（CRT）显示器、液晶显示器（LCD）以及等离子显示器等。后两者是平板式的显示器，由于体积小，功耗低，一般用于笔记本计算机。按照显示内容，显示设备又可以分为字符、图形、图像显示器三种。

PC 机系列中所用的显示系统有以下几种：

• MDA（monochrome display adapter）：单色字符显示系统的适配器，采用 9×14 点阵的字符，一屏可以显示 80 列 × 25 行字符，对应的分辨率为 720×350 像素。

• CGA（color granhics adapter）：彩色图形适配器，有字符和图形两种方式。在字符方式下，每屏 25 行×80 列，每个字符有 16 种颜色。在图形方式下分辨率有两种：320 ×200 / 4 色，640×200 / 2 色。

• EGA（enhanced graphics adapter）：增强型图形适配器。字符方式下每屏 25 行×80 列，每个字符有 64 种颜色。在图形方式下，分辨率可达 640×350 / 16 色。改进型的 EGA 分辨率可达 640×480 / 16 色。

• VGA（video graphics adamer）：视频图形适配器。标准的 VGA 显卡分辨率是 640×480 / 16 色，而增强型的 VGA 卡，分辨率可以达到 800×600、960×720、甚至 1024 × 768 / 256 色。
性能参数主要有：

• 分辨率和灰度级：分辨率指显示设备所能表示的像素个数。像素越多则分辨率越高，图像也就越清晰。灰度级是指每一像素点的亮暗层次级别，或者可以显示的颜色数目，其值越高，图像层次越清楚逼真。若用 8 位来表示一个像素，则其可以有 256 级灰度或颜色。灰度级取决显示器本身的性能以及其刷新存储器的大小。

• 刷新率：因为 CRT 是通过电子来打击荧光屏而显示图像的，电子束扫过之后，几十毫秒之后，其发光亮度会逐渐减弱以至消失。为了得到稳定的图像，必须在其消失之前对其进行刷新（即再次显示）。一般每秒刷新 50 帧图像。刷新率越高，图像越稳定。否则会有抖动感。为了给刷新提供信号，必须把图像存储起来。

用来存储该图像的存储器为刷新存储器，又称为帧存储器。其大小和分辨率、灰度级有关。如对于 640×480，65536 种颜色（2^{16}）的图像，则需要
640× 480 ×16=614.4KB

②打印设备

打印机主要可以分为击打式和非击打式。击打式打印机是利用机械动作，使之与色带和打印纸相撞而印出打印的内容。又可以分为活字式打印机和点阵式打印机。活字式打印机是把每一个字刻在印字机构上，有球形、菊花瓣形、鼓轮形等。活字式打印机打印汉字较困难。点阵式打印机是利用打印钢针组成的点阵来打印的。其价格低，结构简单，可以打印字符，汉字以及图形等等。根据打印头上的钢针数目一般有 9 针的和 24 针两种。

非击打式打印机是利用光、电、磁、喷墨等物理、化学方法把内容打印出来。由于其体积小，噪声低，印刷清晰，速度快。而且随着技术发展，价格也逐渐下跌，大有取代击打式打印机的趋势。常见的非击打式打印机主要有喷墨式和激光打印机。

喷墨式打印机是把特制的墨水用精制的喷头喷洒到纸面上以印出各种内容。激光打印机是利用电子照相原理以及静电复印原理进行输出的。激光打印机的印字记录装置上均匀涂有静电电荷，被激光打中的地方产生了放电现象--形成静电印刷板；再利用静电复印原理，把有内容的地方吸附上碳粉印刷到普通纸上。最后把纸张加热定影，然后输出。

2.I / O 系统的工作方式

I / O 系统主要有 3 种方式来与主机交换数据。

（1）程序控制方式（programmed I / O）

数据的输入 / 输出完全由 CPU 控制，在整个 I / O 过程中 CPU 必须等待而不能做其他工作，无法充分发挥 CPU 高速处理能力。

（2）程序中断方式（interrupt-driven I / O）

I / O 系统以中断的方式通知 CPU 开始传递数据，勿需 CPU 主动查询和等待外设，从而把 CPU 解放出来做其他工作，提高了 CPU 的利用效率。

由程序控制 I / O 方法的主要缺点在于 CPU 必须等待 I / O 系统完成数据传输任务，在此期间 CPU 需定期地查询 I / O 系统的状态，以确认传输是否完成。因此整个系统的性能严重下降。为了克服该缺陷，把中断机制引入到 I / O 传输过程中来。CPU 利用中断方式完成数据的输入 / 输出，当 I / O 系统与外设交换数据时，CPU 勿需等待也不必去查询 I / O 系统的状态，而可以抽身出来处理其他任务。当 I / O 系统完成了数据传输后则以中断信号通知 CPU。CPU 然后保存正在执行程序的现场，转入 I / O 中断服务程序完成与 I / O 系统的数据交换。然后再

返回原主程序继续执行。与程控 I／O 相比，中断方式因为 CPU 勿需等待而提高了效率。

（3）DMA 方式（direct memory access）

中断法虽然比程序控制法更加有效（也能快速响应外设的要求），但两者都是由软件来完成数据的传输，难以胜任高速的数传需求。并且在主存和 I／O 模块之间进行数据交换时仍需由 CPU 控制，并且每一数据都需要经过 CPU 中转。因此不仅影响了 CPU 的工作效率也降低了 I／O 系统的数据传输率。

在 DMA 方式下,CPU 只是在数据传输前和完成后才介入,而数据的传输过程由 DMA 控制器（DMAC）来管理，勿需 CPU 参与。数据直接写入或读出主存储器，不再经过 CPU 中转。

DMA 方法使用 DMAC 来控制和管理数据传输。DMAC 和 CPU 共享系统总线，并且具有独立访问存储器的能力。在进行 DMA 时，CPU 放弃对系统总线的控制而由 DMAC 控制系统总线；由 DMAC 提供存储器地址信号及必须的读写控制信号，实现外设与存储器之间的数据交换。

DMA 获取系统总线的 3 种方式：

• 暂停方式：DMAC 请求 CPU 让出系统总线，由 DMAC 使用，直到一组数据全部传送完毕后，才把总线控制权还给 CPU。

• 周期窃取方式：请求 CPU 进入空闲态暂时放弃总线，插入一个 DMA 周期，传送完一个字后，把总线还给 CPU。相当于 CPU 暂停了一个总线周期。

• 共享方式：在 CPU 不使用总线的时候由 DMAC 进行 DMA 传输。

3.设备接口（external interface）

设备接口是 I／O 系统与外部设备进行数据交换的桥梁，用以适应各种千差万别的外部设备。

（1）接口的类型

根据外部设备与 I／O 系统交换数据的方式可以分为两种：串行和并行接口。串行接口一次只能传送 1 个 bit 信息，而并行接口一次就可传送多位 bit（如 8hit，10bit，16bit，32bit 等）。并行接口数据传输率高，控制简单，通常用于高速数据通道接口；但是所需连线很多，不适于制复杂而且传输速度较慢。

（2）常见的磁盘设备接口

①ST506 接口

该接口主要用于使用同步电机作定位机构的温盘，结构简单，只完成磁盘信息的读写放大，把数据的编码解码、数据的格式转换等功能都留给由 I / O 系统处理。其传输速率为 5Mbps～7Mbps，最多可支持 2 个硬盘，最大支持盘空间为 150MB。

②ESDI（enhanced small device interface）接口

一种通用的标准接口，不仅适用于小型温盘，还适用于磁带机和光盘存储器。该接口除了完成信息的读写放大外，还要完成数据的编码解码。数据传输速率 5Mbps～10Mbps，最多可支持 4 个硬盘，硬盘空间最大可达 600MB。

③IDE（integrated drive electronics）接口

该接口是微机中常用的接口标准，分为普通 IDE 和增强型 IDE 接口。

• 普通 IDE

数据传输率不超过 1.5MB/s，数据传输宽度为 8 位；最多可连接 4 个 IDE 设备；每个 IDE 硬盘容量不超过 528MB。

• EIDE 接口

数据传输率可达 12MB / s～18MB/s，数据传输宽度 32 位设备；每个 IDE 硬盘容量可超过 528MB。

④SCSI（small computer system Interface）接口

自从 1984 年 SCSI 接口应用于 Macintosh 机后，现在它不仅应用于 Macintosh 机，而且还广泛应用于 PC 兼容机以及许多的工作站上作为一种系统级的设备接口。SCSI 是并行接口，其数据宽度有 8hit，16bit 和 32bit 三种，它是大容量存储设备、音频设备和 CD-ROM 驱动器的一种标准接口。SCSI 接口通常被看作是一种总线，用于连接多个外设，这些 SCSI 设备以雏菊链（daisy－chained）形式接入，并被分配给一个惟一的 ID 号（0～6），其中 7 号分配给 SCSI 控制器。某些 SCSI 控制器可以提供多达 35 个 SCSI 通道。SCSI 设备彼此独立运作，相互之间可以交换数据，也可以和主机进行交换，数据以分组消息（messase packet）的形式进行传输。

（3）微机上新型的常用接口

①通用串行总线（Universal Serial Bus，USB）

这是由 Intel 公司开发的一种高速串行总线端口，通过它来连接外部设备，由于可以串行连接多个设备，可减少对宝贵系统资源的占用。目前在 PC 机和便携机上一般都装有 USB 口。

②个人计算机小型通信接口适配器（Personal Computer Miniature Communications Interface Adapter,PCMCIA）

目前在便携机上已普遍提供该插槽，又称 PC 卡插槽。在该槽内可插入标准尺寸的约一张信用卡大小的各种插卡，如闪存、SRAM、Modem 和 SCSI 等，从而使其可用性和扩展性大大增强。

1.5 指令系统

CPU 所完成的操作是由其执行的指令来决定的，这些指令被称为机器指令。CPU 所能执行的所有机器指令的集合称为该 CPU 的指令系统（又称指令集）。从计算机设计人员的角度而言，指令系统描述了 CPU 的基本功能，设计 CPU 很大程度上就是实现其指令系统；从使用机器语言编程的用户角度而言，其面对的就是 CPU 的寄存器、存储器、机器直接支持的各种数据类型以及 ALU 等计算机系统的主要部件。

1．5．1 机器指令的格式

1．指令格式

每一条指令都必须包含相应的信息以使 CPU 能够执行。通常机器指令由以下 4 部分组成：操作码、源操作数、目的操作数以及下一条指令的地址。操作码指明该指令所要执行的操作（例如加法运算或者分支跳转等）；源操作数是该操作的输入数据；目的操作数则是操作的输出数据。而下一条指令地址则通知 CPU 到该地址去取下一条将执行的指令，大多数情况下，下一条指令地址是以隐含形式给出，也即紧接当前正当执行的指令之后。

在计算机内部，机器指令以二进制位形式表示和存放。但是为了方便描述和讨论，通常以助记符号的形式来表示机器指令。例如，我们用 ADD AX，BX 来表示一条机器指令，ADD 是操作码的助记符，AX 和 BX 是寄存器操作数助记符，BX 是源操作数，AX 既是源操作数，又是目的操作数。这条指令完成把寄存器 AX 和 BX 里的数据相加，结果存放到 AX 寄存器中。

2．指令操作码

CPU 的指令系统可能包括几十到数百条机器指令，每一条指令都对应着一个相应的操作码。指令操作码通常有两种编码格式：固定格式和可变格式。

固定格式的操作码的长度是固定不变的，广泛应用于大、中型机和超级小型机，有助于简化硬件电路和缩短指令的译码时间。一个长度为 n 位的操作码则允许该指令系统中包含 2n 种指令。

可变格式操作码的长度不是固定的，且操作码不再集中于指令的一个字段中而是分散扩展到各个不同的字段。使用统计方法，给经常出现的指令分配给短的操作码，出现频率低的指令相应其操作码长度也较长。这样可以有效地压缩程序中操作码的平均长度。这种编码格式常用于字长较短的小型机和微型机中。操作码的长度可变，根据指令在程序中出现的概率，优化分派各种操作码不同的长度，可

以增加指令字所能表示的操作信息，缩短指令的平均长度，减少程序的总长度。但是这样也增加了指令译码的难度，使控制器电路更加复杂。

3．操作数

在指令的执行过程中，CPU 把指令取至指令寄存器中，对操作码进行译码以确定所要执行的操作，同时还要根据指令中给出的操作数地址（或操作数）去读取源操作数，并把执行的结果保存回目的操作数地址指定的存储单元。在本章中，我们以 X 表示地址，用（X）表示存放在该地址单元中的数据。

根据指令中可能含有的操作数地址的个数来划分，通常有 0 操作数、1 操作数、2 操作数、3 操作数等指令格式。

（1）0 操作数（0 地址）指令

其格式为：OP

指令中只有操作码，没有操作数。用于不需要操作数的指令如空操作指令、停机指令等，以及用于默认操作数地址的指令中。如在堆栈操作中，操作数默认存放在堆栈栈顶的两个存储单元中。

（2）1 操作数（1 地址）指令

其格式为：OP A

指令中给出了一个操作数地址，用作源操作数。至于目的操作数则可能有两种情形：

• 隐含操作数：在早期的微型机中，第二操作数隐含在 CPU 的一个特殊寄存器（累加器 AC）中。指令的执行意味着：（A）OP（AC）→AC。

• A 既是目的操作数又是源操作数地址。指令的执行则代表：OP（A）→A。例如常用的加 1、减 1 指令：INC AX 把 AX 寄存器中的数加卫后再放回 AX 中。

（3）2 操作数（2 地址）指令

指令格式为：OP A，B

指令中给出了两个操作数地址，源操作数地址为 A 和 B，至于目的地址则由指令系统规定是 A 或者是 B。若规定目的地址为地测指令的含义为：（A）OP（B）→A。

（4）3 操作数（3 地址）指令

指令格式：OP A，B，C

指令中给出了 3 个操作数的地址，一般规定 B 和 C 用作源操作数地址，A 用作目的地址。指令表示：（B）OP（C）→A

（5）多操作数（地址）指令

在某些高性能计算机中，某些指令中包含了多个操作数地址以实现成批数据处理，如向量运算、矩阵运算，字符串处理等。

4．数据格式

（1）数据种类

计算机指令处理的基本数据有 4 种：地址数据、数值数据、字符数据和逻辑数据。

①地址数据

指令中的地址信息实际上也是一种数据，常常对其执行某种运算以求得新的操作数地址。从这种意义上看，地址数据其实就是无符号的整数。

②数值数据

常用的数值数据有定点数（整数）、浮点数（实数）以及十进制数等。因为计算机中用于表示数值数据的位数是有限的而且在用浮点法表示数据时固有的精度损失，所以在计算机中进行运算时，不可避免地会遇上结果的溢出（上溢出或下溢出）或者舍入（rounding），需要正确地处理这些情况。

尽管计算机内部适用于二进制方式表示数据，但对于人而言还是习惯于用十进制进行运算。所以在用计算机处理问题时就需要进行两次转换：十进制输入，然后转变为二进制进行处理，最后再转变为十进制输出。在某些应用中，大量的处理在于数据的输入 / 输出，而相对来说计算只占一小部分且较简单，在这种情况下，更适合于用十进制数进行处理，以提高效率。

③字符（串）数据

我们需要交给计算机处理的数据除了用以计算的数值数据外，还有用于文字信息处理的字符数据。这些字符数据不能直接用于基于二进制的计算机中，因此必须经过某种形式的编码（如 ASCII 码、EBCDIC 码）用二进制位串来表示字符。

④逻辑（布尔）数据

一般情况下，数据是以其存储单位 addressable unit（可寻址单位）作为一个整体来处理的，例如字节（byte）、半字（half word）、字（word）和双字（double word）等。但是在某些情况下，需要对数据按二进制位（bit）逐位进行处理--这些数据被看作为逻辑数据。

实际上在计算机内部，各种数据都是以二进制位的形式存放。具体的数据类型其实是由在数据上执行的操作或运算来决定的。

（2）数据的组织

计算机的存储器一般按字节进行组织，但其指令系统可能支持对半字、字或多字的运算。为了便于硬件实现，一般要求多字节数据对准边界（alignment）。若数据位数不足时则以空白字节填充。这样可以减少对存储器的访问次数。多字节数据涉及的另一个问题则是其高、低位字节的排列顺序。在不同类型的计算机中其顺序也不同，有的是低位字节存放于低地址，而另一个方案则把高位字节放在低地址。 1．5．2 指令的种类和功能

计算机的指令系统不仅与计算机的硬件结构紧密相关，而且一个精心设计的指令系统对提高计算机的性能也有着深刻的影响。如何设计和选取指令，以实现指令系统一直是计算机系统设计中的核心问题。因为各种指令种类繁多，指令系统如何取舍关系重大。
尽管不同的计算机的指令系统各不相同，但是基本上所有的计算机都包括了如下几种类型的指令：

- Data Transfer：数据传送类；
- Arithmetic：算术运算类；
- Logleal：逻辑类；
- Conversion：数据变换类；
- I／O：输入／输出类；
- System Control：系统控制类；
- Transfer of Control：控制权转移类。

1．数据传送类指令

这类指令用以实现把源地址的数据传送到目的地址去，一次可以传送一个或者一批数据。有的指令系统中还包括交换指令，可以把源操作数和目的操作数进行互换。
数据可以在寄存器与寄存器之间、寄存器与存储器单元之间以及存储器单元与存储器单元之间传递。

常用的传送指令有。

- MOVE：在源和目的地址间传递数据；
- IOAD：从存储器单元向 CPU 传递数据；

• STORE：从 CPU 向存储器单元传递数据；

• EXCHANGE：把源操作数和目的操作数进行互换；

• PUSH：堆栈人栈指令，把源操作数送入堆栈的栈顶；

• POP：堆栈出栈指令，把栈顶数据送入目的单元。

2．算术运算类指令

大多数计算机都提供了基本的算术运算指令，例如加减、求补、比较以及乘、除指令，一般而言是对带符号整数进行运算。但是为了提高计算机的性能，某些用于科学计算的计算机往往设置有浮点运算指令，对浮点数据（例如 32 位的单精度数、64 位的双精度数）进行运算。还有的指令系统中带有十进制运算指令，能够对十进制数进行运算，从而提高数据交换频繁的应用系统的性能。

3．逻辑类指令

逻辑类指令对操作数逐 bit 位进行操作，可以分为基于布尔运算的逻辑运算指令和移位操作指令。

（1）逻辑运算指令

主要实现非（NOT）、与（AND）、或（OR）以及异或（XOR）等逻辑运算功能。某些计算机的指令系统中还包括了位操作指令，如位测试、位清除等。

（2）移位操作类指令

通常在计算机系统中提供了 3 类移位指令：

①算术移位

算术移位指令是对带符号数进行移位，其执行过程如图 1．14 所示：左移时从最低位依次向最高位移动，最低位补 0，最高位移入到"进位"位 C 中。而右移时则从最高位向最低位依序移动，最低位进入到"进位"位 C 中，而最高位（即符号位）保持不变--拷贝自身。

②逻辑移位

逻辑移位指令对无符号数进行移位。其左移操作与算术左移相同，但逻辑右移不同于算术右移：其最高位不再拷贝自身而是用 0 补充（见图 1．15）。

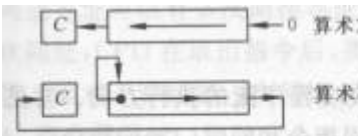


图 1.14 算术移位操作

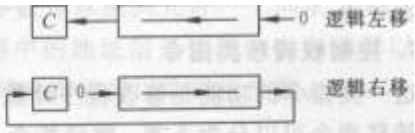


图 1.15 逻辑移位操作

③循环移位

循环移位指令则又可以分为不带"进位"位C的小循环和带"进位"位C一起循环的大循环。如图 1．16 所示。

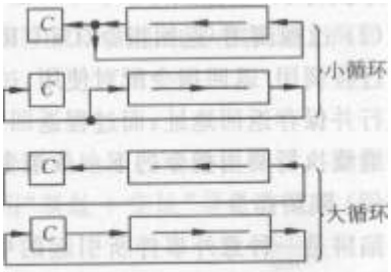


图 1.16 循环移位操作

算术／逻辑移位指令可以实现对带符号数／无符号数的简单乘除运算。例如左移 n 位相当于乘以 2n 而右移 n 位则相当于整除以 2n。

4．数据变换类指令

这一类指令用于对于操作数的格式进行变换。例如把十进制格式的数据转换为二进制。

5．输入／输出类指令

计算机经常需要和外设进行数据交换。有的计算机把外部设备和存储器统一编址，因而任何对存储器读写的指令均可用来访问外设，其指令系统中没有专门的 I／O 指令。而有的计算机中外部设备和存储器拥有相互独立的存储空间，其指令系统中设有专门用于访问外部设备的 I／O 指令。一般 I／O 指令有以下几种：

• Read（Input）：从指定的外部设备读取数据；

• Write（Output）：向指定的外部设备写入数据；

• Start I／O：用以向 I／O 处理机传送命令以初始化 I／O 操作；

• Test I／O：读取并测试 I／O 系统的状态信息。

其指令格式一般如下：

OP	REG	A

OP 是操作码，REG 为用于和外部设备进行数据交换的寄存器，而 A 用于指定外部设备的 I／O 地址。

6．系统控制类指令

系统控制类指令是只能由操作系统等系统软件使用的特权指令，主要用来分配和管理系统资源，如改变系统的工作方式、检测用户访问权限、修改虚拟存储器系统中的段表和页表等等。

7．控制权转移类指令

这一类指令的功能是修改程序计数器的值，从而改变程序流的执行方向。常见的控制权转移指令可以分为 3 类：跳转指令、过程调用／返回指令和陷阱（Trap）指令。

（1）跳转指令

跳转指令又可以分为无条件跳转指令和条件跳转指令。无条件跳转指令直接使程序跳到指令中所标明的目的地址去执行程序，而条件跳转指令则根据指令中所规定的条件成立与否来决定程序的流向。若条件成立则跳到指令中的目的地址处执行，条件不成立就顺序执行下一条指令。

（2）过程调用／返回指令（Call／Return）

过程调用／返回指令配对使用：过程调用指令使 CPU 从一个程序段转到另一个程序段执行并保存返回地址；而过程返回指令则根据调用指令所保存的返回地址返回到原程序段继续执行调用指令的下一条指令。返回地址通常保存在堆栈中。

（3）陷阱指令

陷阱是一种意外事件所引起的中断。通常陷断指令是隐含的，不提供给用户使用。当意外事件引发中断后，由 CPU 自动产生和执行。
但也有某些指令系统提供了用户使用的陷断指令，使用户可以用它来进行系统调用，从而可以利用由操作系统提供的各项系统功能。

1．5．3 指令的寻址方式

寻址方式就是用来确定操作数地址和下一条要执行的指令地址的方法。常用的指令寻址方式有以下 7 种：

1．立即数寻址方式

指令中直接给出操作数，取指时操作码和操作数同时读出，无需再次访问存储器。指令的执行速度快，但是因为操作数包含在指令中而无法修改，只能用于操作数为常数的情况，通常用于定义常数或为变量赋初值。

2．直接寻址

指令的地址码给出了操作数的地址。CPU 需要根据该地址再次访问存储器以读入操作数。

3．间接寻址

指令的地址码给出的是存放操作数的存储单元的地址信息，即所谓"地址的地址"。这种寻址方式又可以分为寄存器间接寻址方式和存储器间接寻址方式。前者使用寄存器来保存操作数地址而后者则把操作数地址保存到另一存储器单元中。

间接寻址可以有多次间接寻址的方式，但大多数计算机只允许一个间址。对于寄存器一次间址，CPU 在取出指令后，要根据该寄存器中的地址信息访问一次存储器以取得操作数。而对于存储器一次间址，CPU 则需访问两次存储器以取得操作数。

4．寄存器寻址

这种方式类似于直接寻址，只不过指令的地址码给出的是存放操作数的寄存器。CPU 无需再次访存，只需读取该寄存器即可取得操作数。而寄存器的读写速度远高于存储器。因而这种寻址方式有助于缩短指令长度和提高指令的执行速度。

5．基址寻址方式

指令中给出操作数地址的偏移量，其基地址存放在一个专用的基址寄存器中，或者也可以存放在由指令中所规定的某个通用寄存器中。操作数的地址由基地址加上偏移量即可求出。

6．变址寻址方式

指令中给出操作数地址的固定部分和存放其可变部分地址的变址寄存器。操作数的实际地址由固定部分加上可变部分即可求出。当使用"基址十变址"寻址方式时，还需要加上基址寄存器中的基地址。

7．相对寻址方式

在该寻址方式下，所需地址等于程序计数器 PC 内的值加上指令中所给出的位移量。因为在现代的计算机中，程序和数据一般分开存放，程序区的内容在程序执行过程中不允许修改，所以通常不用相对寻址方式来确定操作数地址。相对寻址主要用于跳转类指令。
实际上不同的计算机系统中采用的寻址方式也往往各不相同，而且即使是对同一种寻址方式也可能有不同的处理方法。

1．5．4 指令的执行过程

通常一条指令的执行可以分为以下 7 个阶段：

- 按照程序计数器 PC 中的地址从内存中读取指令；
- 对指令译码以确定其所要实现的功能；
- 计算操作数的地址；
- 从该地址读取操作数；
- 执行操作；
- 保存结果；
- 计算下一条要执行的指令的地址，存人程序计数器 PC。

需要说明的不是每一条指令都需要经过以上 7 个阶段，例如采用立即数寻址方式的指令就无需计算操作数地址以及再从存储器读取操作数这两个阶段。

第 2 章 语言处理程序基础知识

程序语言可以划分为低级语言和高级语言两大类。低级语言又称面向机器语言，它是特定的计算机系统所固有的语言。用机器语言进行设计，需要对机器结构有较多的了解。用机器语言编制出来的程序可读性很差，程序难以修改和维护。为了提高程序设计效率，人们考虑用有助记忆的符号来表示机器指令中的操作码和操作数，例如用 ADD 表示加法，SUB 表示减法等，这就是汇编语言。虽然用汇编语言来编制程序，使编制程序的效率和程序的可读性都提高了；然而，汇编语言是一种和计算机的机器语言十分接近的语言，它的书写格式在很大的程度上取决于特定计算机的机器指令，这是一种低级语言，这对于人们抽象思维和交流十分不便。在这个基础上，高级语言就发展起来了。目前已有许多高级语言在流行，如 FORTRAN，COBOL，PASCAL，C，C++等。这类语言与人们的自然语言比较接近，大大提高了程序设计的效率，便于人们用这类语言进行交流。

今天的计算机仍然只能理解和执行机器语言。程序语言的引入意味着：必须有一个程序，其任务是使机器能够理解用某一程序语言书写的用户程序，担负这一任务的程序称为“语言处理程序”，它可以分为两大类：解释程序和翻译程序。

解释程序接受所输入的用程序语言（称为源语言）书写的程序（称为源程序），然后直接解释执行源程序（或源程序的内部形式）。翻译程序则接受某种语言程序（源语言程序）后，将它改造成另一种语言程序（称为目标语言程序），而后

者与前者在逻辑上是等价的。如果源语言是汇编语言而目标语言是机器语言，则这种翻译程序称为汇编程序。如果源语言是高级语言。而目标语言是机器语言，则这种翻译程序称为汇编程序。如果源语言是高级语言，而目标语言是低级语言（汇编语言或机器语言），则这种翻译程序称为编译程序。

2.1 程序语言基础知识

2.1.1 概述

程序语言已经历了 40 多年的发展，其间人们提出并完善了许多程序语言的概念。程序语言具有交流算法和计算机实现的两重目的，现在程序语言种类繁多，它们在应用上各有不同的侧重面。

FORTRAN 是第一个被广泛应用进行科学计算的高级语言。一个 FORTRAN 程序由一个主程序或一个主程序与若干个子程序组成。主程序及每一个子程序都分别是独立的程序单位，称为一个程序模块。在 FORTRAN 中，子程序是实现模块化的有效途径。FORTRAN 的结构特别简单，除了输入和输出部分处，几乎所有的 FORTRAN 成分都可以用硬件结构直接实现，从而使执行效率相当高。

ALOGOL 是另一个早期研制出来的高级语言。它有严格的文法规则，用巴科斯范式 BNF 来描述语言的文法。ALGOL 是一个分程序结构的语言。一般来就，一个 ALGOL 程序本身就是一个分程序。每个分程序由括号 begin…end 括起来，以说明它的范围和它的管辖名字的作用域。分程序的结构可以是嵌套的，也就是说，分程序内可以含有别的分程序。过程也可以看成是一个分程序，这个分程序可以在别的分程序中被调用。同一名字在不同的分程序中可以代表完全不同的实体。如果一个名字在若干层嵌套分程序中多次被说明，则程序该名字的使用由离使用点最近的内层说明决定，即“最近嵌套原则”。此外，ALGOL 还提供了数组的动态说明和过程的递归调用。

分程序结构的主要优点是：可以非常有效地使用存储器。因为一个分程序只有在执行时才需要数据空间，执行后所占用的空间被释放。由于分程序结构的嵌套性，因此，可用一个栈作为整个程序运行时的数据空间，这种存储管理方式非常方便。

COBOL 是一种面向事务处理的高级语言。COBOL 把新的概念引入程序设计语言。例如，程序中数据的描述（在 COBOL 的数据部）与具体计算机无关，这一点后来又对数据库管理系统析发展产生重大影响。由于政府和工业界的合作，COBOL 被人们广泛接受，至今仍在事务处理领域得到广泛使用。

PASCAL 语言提供为数不多然而又相当紧凑的机制，使得这个语言具有相当强的表达能力，它的数据结构功能，尤其是用户自定义的数据类型以及该语言的公理化定义，使得它在高级语言的领域内具有很大吸引力。此后，PASCAL 语言中添加了并发控制结构，产生了并发 PASCAL。在 PASCLA 中分程序和过程这两个概念

合二为一，统一为过程。而一个 PASCAL 程序本身可看成是一个由操作系统所调用的过程。PASCA 过程可以是嵌套和递归的。

C 是 20 世纪 70 年代发展起来的一种通用程序设计语言。字符、整数和浮点数是 C 的基本数据对象，用户可以用指针、数组、结构和联合等建立新的数据类型。C 是一种较低级语言，它提供了指针和地址操作能力。C 提供书写结构良好的程序所需的控制结构。C 与 UNIX 操作系统紧密相关，UNIX 操作系统及其上的许多软件都是用 C 编写的。

C++是在 C 语言的基础上发展起来的，与 C 兼容。在 C++中，最主要的是增加了类功能，使它成为面向对象的程序设计语言。一般认为，面向对象程序语言至少包含下面一些概念：

（1）对象：对象是人们要进行研究的任何事物，它具有状态（用数据来描述）和操作(用来改变对象的状态)。面向对象语言把状态和操作封装于对象体之中,并提供一种访问机制,使对象的”私有数据”仅能由这个对象的操作来执行。用户只能通过向允许公开的操作提出要求（消息），才能查询和修改对象的状态。这样，对象状态的具体表示和操作的具体实现都是隐蔽的。

（2）类：类是面向对象语言必须提供的由用户定义的数据类型，它将具有相同状态、操作和访问机制的多个对象抽象成一个对象类。在定义了类以后，属于这种类的一个对象叫作类实例或类对象。一个类的定义应包括类名、类的说明和类的实现。

（3）继承：继承是面向对象语言的另一个必备要素。类与类之间可以组成继承层次，一个类的定义（称为子类）可以定义在另一个已定义类（称为父类）的基础上。子类可以继承父类中的属性和操作，也可以定义自己的属性和操作；从而使内部表示上有差异的对象可以共享与它们结构中的共同部分有关的操作,达到代码重用的目的。

逻辑型语言是一类以形式逻辑为基础的语言。PROLOG 是这类语言的代表。PROLOG 建立在关系理论和一阶谓词理论基础上。逻辑程序设计的实质是实现关系。以值集合 S 和 T 上的二元关系 R 为例，关系 R 实现后，我们可以询问：

- 已知 a 和 b,确定 R(a,b)是否成立；
- 已知 a,求所有使 R(a,y)成立的 y；
- 已知 b,求所有使 R(x,b)成立的 x；
- 求使 R（x，y）成立的所有 x 和 y。

这些查询清楚地显示了逻辑型程序设计的特点,它具有和传统的命令型程序设计完全不同的风格。PROLOG 程序由一些俗称事实和规则的 Horn 子句组成。Horn 子句形如“ A_0:- A_1，…， A_n。”,其中 A_i为形如 R_i（…）的断言，而 R_i

是关系名。该子句表示：如果 A_1,…, A_n均为真（成立），则 A_0也为真。当 n=0 时，Horn 子句变成：“ A_0”，这样的子句被称为事实。

一旦有了事实和规则后，我们就可以提出询问。测试用户询问 A 是否成立采用归结方法：

- 如果程序中包含事实“ A_0”，且 A_0和 A 匹配，则 A 成立。

- 如果包含了 Horn 子句“ A_0:- A_1，…， A_n”，且 A_0和 A 匹配，则 PROLOG 转而测试“ A_1，… A_n”；只有当这些“ A_1，… A_n”都成立时才能断言 A 成立。当某个 A_i失败时，PROLOG 就回溯。所谓回溯，就是放弃和 A_i匹配的那个 Horn 子句，继续测试紧着下面的 Horn 子句。

- 只有当所有可能情况都已穷尽时，才能导出 A 失败。

PROLOG 具有很强的推理功能，适用于书写自动定理证明、专家系统、自然语言理解等问题的程序。

函数程序语言是一类以 λ 演算为基础的语言。LIST 是典型的函数型程序语言。函数是一种对应规则(映射)它使其定义域中每一个值和值域中惟一的值相对应。例如：

```
Plusone[x]:=x+1

Plustwo[x]:=Plusone[Plusone[x]]

fact[n]:=if n=0 then 1 else n*fact[n-1]
```

在第二个函数定义中，使用了函数复合，即将一个函数调用嵌套在另一个函数定义中。在第三个函数定义中，函数被递归定义。由此可见，函数可以被看成是一种程序，其输入就是定义中左边括号中的量。它也有将输入组合起来产生一个规则,组合过程中可以使用其他函数或该函数本身, 这种用函数和表达式建立程序的方法就是函数程序设计。函数型程序设计语言的优点之一是,对表达式中出现的任何函数都可以用其他函数来代替,只要这些函数调用产生相同的值。由于用函数程序设计语言书写的程序是利用自变的值来计算函数的值,它没有副作用。这些特点有助于程序的模块化。

2.1.2 语言提供的数据类型

任何一个程序都可看成是对一些数据及作用于该组数据上的操作的一种说明。不同语言所提供的数据类型不尽相同,本节对 C/C++所提供的数据类型作一概述。

数据是程序操作对象,具有存储类、类型、名称、作用域和生存期等属性，使用时要为它分配内存空间。数据名称由用户通过标识符命名；类型说明数据占用内存的大小和存放形式；存储类说明数据在内存中的位置和自下而上期；作用域说明数据可以使用的范围；生存期说明数据占用内存的时间。

1. 数据分类、数据说明

数据从不同角度可分成不同的类别。按数据的作用域大小，可分为全局量和局部量；按生存期可分为自动生存期、静态生存期和动态生存期；按程序运行时数据的值是否能改变可分为常量和变量。

数据按类型可分为 4 种：void、标、(scalar)函数和聚合(aggregate)。标量和聚合类型又进一步划分为：

scalar:算术、枚举、指针和引用；

aggregate:数组、结构体、共用体和类。

数据按其构造方式又可分为基本类型和派生类型。基本型是
void,char,int ,float,double 和枚举类型，以及它们的变种
short ,long,signed 和 unsigned。派生类型包括指针、引用、数组、函数、结构体、共用体和类。其中，最后三种为用户类型（见图 2。1）。

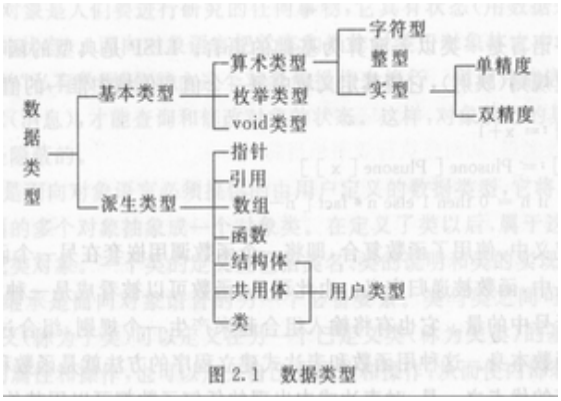


图 2.1 数据类型

数据必须先说明，后使用，这是一条基本原则。

数据说明指定数据的存储类、类型和名称，有时还要初始化，在说明数据的同时，为它赋初值。数据在说明时一般为它分配内存。数据说明的形式如下：

存储类 类型 名称

其中，名称由用户为数据命名；类型是上面提到的基本类型、指针、引用或用户定义的类型；存储类用下列 4 个关键字之一说明：

auto,regist,static,extern

存储类可以默认。对函数，默认值是；extern 对其他数据，默认值是 auto。

2。 算术数据类型

算术变量是程序中经常使用的最简单的数据，它的值是一个数或字符。数值是有范围和精度要求的，同一程序在不同计算机上运行可能会产生不同的结果。

如果算术表达式中的操作数据类型不同，则要进行隐式类型转换，使表达式中的数据类型相同。其转换规则为：第一，表达式中有 char,short 和 enum（枚举）类型的数据，自动将它们转换成 int 类型；第二，把表达式中不同类型的数据转换成精度最高、占用内存最多的那个数据的类型。例如，表达式中有 char,int,float 和 double 类型的数据时，都转换成 double 类型。

在类型转换中，将精度低、占用内存少的数据类型转换为精度高、占用内存多的数据类型时，仅仅改变类型，不改变它的值；反之，仅改变类型，也可能改变其值,如在赋值表达式中存在的自动转换和强制转换。类型转换要占用一定的时间，编程时应尽量避免类型转换。

3. 枚举数据类型

枚举是为一组整数值提供便于记忆的标识符，形式如下：

enum 枚举名｛枚举常量表｝;

它说明枚举名是一种数据类型，简称枚举。花括号中的枚举常量表是一组用逗号分隔的表示整型数值的标识符。例如：

enum days ｛SUN,MON,TUE,WED,THU,FRI,SAT｝;

枚举常量是符号常量，它的值 int,char 或 unsigned。一般情况下，第一个枚举常量的值是 0，后面的按序递增 1。例如，days 中各枚举常量的值分别为：

0,1,2,3,4,5,6

枚举常量可以定义时由用户赋值，值也可以相等；后面的枚举常量的值也可用它前面的枚举常量表示。例如：

enum colour｛WHITE=1,RED=3,BLUE,GREY=WHITE+6｝;

说明枚举常量 BLUE=4，GREY=7。

枚举常量被定义后，其值在程序中不变，只可以使用，不可以改变。

枚举常量用便于记忆的符号表示整数,比直接使用数字直观易读,便于类型检查。

定义了一个枚举类型后，就可以用它说明枚举变量。枚举变量的值只能是相应的类型的枚举常量。例如： days anyday;

说明 anyday 是 day 类型的枚举变量，它的值只能是 day 定义中某个枚举常量的值。

通常枚举变量有与整型变量一样的使用方法。例如：

```
anyday=MON;              /*合法*/
anyday=1;                /*编译时有警告性错误*/
anyday=(days)1;          /*合法，先进行类型转换*/
int i=MON;               /*与 i=1 一样*/
int j=anyday;            /*与 j=1 一样*/
```

4。指针数据类型

指针是对象的地址,指针变量是存放某对象地址的变量。指针变量的说明形式为：

类型 *标识符;

它说明“标识符” 是一个指针变量，它的值是一个对象的地址（指针），这个对象的数据类型为“类型”。

指针变量的值如果是 0，则称这个指针变量是空指针。空指针不指向任何变量；大 C 和 C++中，它的值 0 在 stdio。h 文件中定义为 NULL。一个指针变量说明为空指针与指针变量不赋值是不同的。例如：

```
int *p=0;  /*P 是一个空指针，它不指向任何变量*/

int *q;    /*q 的值是不确定的*/
```

应该注意，指针变量和它所指的对象是两种不同的数据，在运算时有自己的规则和限制。指针变量有专门的运算符，指针运算有赋值运算、关系运算和加减运算等。指针变量运算符有取地址运算符&和取内容运算符*,它们是单目运算符。指针变量加减一个整数 n,使指针向前或向后移动 n 个元素,移动的字节数等于该元素所占的内存字节数乘以 n。指针变量加减一个整数，值仍是内存中数据的地址；如果这个值超过数据在内存中的有效地址范围，程序运行结果将难以预料。两个指针变量指向同一类对象时，可以相减；其相减折差表示指针之间这种对象元素的个数，两个指针变量相减常用于数组或字符串。两个指针变量如果指向同一对象，可以对它们进行大小比较。

例如：

```
Int i=3, *p=&I,j
int *pl=p;                    /*p1 和 p 都指向 i*/
*pl =*pl+5;                   /*与 i=i+5 一样*/
j=*p++;                       /*相当于 j=*p; p++; */
```

用关键字 void 说明的指针变量叫通用指针变量，它可以存储各种类型指型指针变量的值，但不能用它取所指类型的数据。例如：

```
int i=3, *ip=&i;      /*ip 指向 i*/

void * vp;            /*vp 是一个通用指针*/

vp=ip;                /*把 ip 的值（变量 i 的地址）赋给 vp*/
```

5.数组

数组是一组类型相同的有序数据。数组用数组名、数据元素的类型和维数表来说明。一维数组的说明形式为：

存储类 类型 数组名[整型常量表达式]；

数组说明后，其数组元素的形式为：

数组名[下标表达式]

其中，下标表达式可以含有常量、变量，但它的值应为整型。C 和 C++不对数组进行边界检查，编程时，应保证数组说明能存下所需数据，并进行边界检查，防止数组越界。

例如：

```
int a [10];                   /*a 是一个整形数组,下标的变化范围为 0～9*/
float r[20]={0。2,3。1};      /*r 是一个实型数组，并对前两个元素赋*/
                              /*初值，即 r[0]=0。2,r[1]=3。1*/
int x []={1，2，3，4};        /*整型数组 x 有 4 个元素，它们的初值依次数 1，2，3，4*/
x=3*r[i+3];                   /*数组元素的使用和变量相同*/
```

多维数组有多个下标。例如：

```
int mat [2][3];       /*mat 是一个整型二维数组*/

char ch[4][3][5];     /*ch 是一个字符型三维数组*/
```

多维数组在内存中按行存放，后面的下标先变，前面的下标后变。例如 mtn 数组在内存中的存放顺序是：

mat[0][0],mat[0][1],mat[0][2],

mat[1][0],mat[1][1],mat[1][2]。

多维数组也可在说明时初始化。例如：

```
int mat [2][3]={ { 1,2,3 }, { 4,5,6 } };

int mat [2][3]={ 1,2,3,4,5,6 };        /*与上式相同*/

int a [3][4]={ { 2 }, { 5 }, { 8 } };  /*仅对第 1 列赋初值*/

int b [3][4]={ { 1 }, { }, { 9 } }     /*仅对第 1 行和第 3 行的第一个元素赋
                                         初值*/
```

6.函数

函数是程序模块的主要成分，它是一段具有独立功能的程序。程序中有关函数的部分有：函数说明和函数调用。函数说明描述了函数的功能，功能的实现通过函数调用来实现。大程序按功能划分成许多功能单一、层次分明的子模块，每个子模块的功能用函数描述和实现。

函数是一种派生类型的数据，和其他数据一样，函数必须先说明，后调用。函数说明包括函数定义和引用说明两种情况。一个函数只能定义一次，它指明函数的原型和函数体。函数的引用说明仅指明函数的原型，表示这个函数已经在别处定义。

函数分为普通函数和类的成员函数。普通函数分主函数、标准函数和用户函数。一个程序必须有一个主函数，名称为 main。

函数的定义形式如下：

存储类 类型 函数名（形参表）

{ 函数体 }

存储类用关键字 extern 或者 static 说明。默认时为 extern。类型说明函数返回值的数据类型；若无返回值，用关键字 void 说明。一对花括号不能少，它指明函数体的开始和结束。

所有函数的定义都是独立的、平行的。在一个函数定义中不允许定义另外一个函数，但可以对别的函数进行调用或作引用说明；它也可以对自身进行调用，即递归调用。 函数原型的形式如下：

函数 函数名（形参表）；

形参表中的形参名可以省略。函数原型提供了更强的函数检查功能，在编译阶段就能发现函数的返回值的类型以及参数个别数和类型是否正确。

函数名（实参表）

C 和 C++是通过传值方式将实参传递给形参。调用函数和被调用函数之间交换信息的方法主要有两种：一种是由被调用函数把返回值返回给主调函数，另一种是通过参数交换信息。用参数交换信息的方法有三种：赋值调用、赋地址调用和引用参数。例如，排序程序中可用函数 swap 来交换两个反序元素。只写

swap(a,b)

是不够的，这里 swap 定义成

```
void swap (int x,int y)                /*错误*/

{ int temp;

      temp=x;

      x=y;

      y=temp;

}
```

因为是传值调用，swap 无法影响调用它的程序中的参数 a 和 b。为了通过函数调用改变实参值，应采用赋地址调用和引用参数的方式。

```
void swap (int * px,int * py)          /*交换*px 和*py*/

{ int temp;

      temp =*px;

      *px =*py;

      *py=temp;

}
```

则调用函数用 swap(&a,&b)来调用，交换它们的值。

引用是 C++中增加的数据类型，用它作函数参数和返回值，使程序更清楚。例如：

```
void swap (int &x,int &y) /*交换 x 和 y*/

{ int temp;

      temp=x;

      x=y;

      y=x;

}
```

用 swap(a,b)来调用它。在 swap 函数定义中，x 和 y 是两个整形变量的别名，函数调用时，实际上使用的是实参 a 和 b 的地址，因此，可以改变它们的值。

7.结构体数据类型

结构体是由用户定义的数据。它的数据成员可以是任何基本类型、派生类型等标准类型的数据，也可以是用户已经定义的数据。

C++扩充了结构体的概念，它不仅包含数据成员，还包含函数成员。

结构体是由不同类型的数据组成的一种新的数据类型。结构体类型定义的一般形式为：

```
struct 结构体类型名 { 成员表 }；
```

定义了结构体类型后，必须用它说明具体的数据，为这种数据分配内存，进行处理。用结构体类型说明的数据叫结构体对象或结构体实例。例如：

```
struct date {                /*date 是一个结构体类型*/

      int year;

      int month;

      int day;

};

struct date d;               /*d 是 date 类型的一个变量*/
```

结构体对象成员的选择使用成员选择运算符。一个叫直接成员选择符，用小数点“。”表示；另一个叫间接成员选择符，用“－＞”表不。使用形式如下：

结构体对。成员名

结构体指针变量－＞成员名

例如：

```
struct date * p;          /*是一个指向 date 类型的指针变量*/
```

用 p->或(* p)。day 来引用 p 所指对象的 day 成员。结构体变量也可在定义时初始化，方法和数组一样。

8。共用体数据类型

结构体一样，C++中的共用体（又称联合）包括数据成员和成员函数。共用体也是用户定义的数据类型。一个共用体变量的所有数据成员共用同一内存空间，内存大小等于占用内存最多的成员长度。例如：

```
union uname {                    /*uname 是一个共用体类型*/
      int a ;
      char c;
      double d;
      float f;
 }
```

用 uname 说明的一个共用体变量，它的 4 个成员 a,c,d 和 f 使用同一内存，其大小为 8 个字节(double 类型数据占 8 个字节)；其中 a 使用前 2 个字节,c 使用前 1 个字节，d 使用全部 8 个字节，f 使用前 4 个字节。

共有体对象的说明和共用体成员的使用与结构体一样，由于共用体成员使用同一片内存区，在某一时刻，只有一个成员的值有效，它覆盖其余成员。

9.类数据类型

C++扩展了 C 语言中的结构体（struct）和共用体(union)的成员，同时引入一个新的类据类型 class。struct,union 和 class 都用来定义新的数据类型——类。

类是一种由用户定义的新的数据类型，它描述了具有相同结构和行为的所有对象组成的集合。属于同一类的对象具有相同的行为，类的接口规定了对用户公开的操作，类的体则定义了这些操作的具体实现，其细节对外不公开，从而保证了类的接口是一种抽象。

类定义说明类的种类、名称、成员和存取规则。一般形式下：

key 类名｛成员表｝

其中，key 是一个关键字，它是 class,struct,union 三者之一。成员表用花括号括住，被封装在类体之中，它包括数据成员、成员函数及存取规则。成员表可以省略，这样的类叫空类。

类成员列表在类体之中，具有一定的存取规则，这些规则规定了存取类成员的权力：

（1）private(私有的)：说明类的成员是私有的，只能被该类的成员函数和友元函数（用 friend 关键字表明，它不是类的成员函数）访问，外界不能直接访问它。

（2）public(公有的)：说明类的成员是公有的，它不仅可以被该类的成员函数访问，而且还可以被外界访问。

（3）protected(被保护的)：说明类的成员为被保护的，它只能被该类的成员函数、友元函数和该类的派生类的成员函数访问。

类的成员函数的说明有两种方法：类内定义和类外定义。在类外定义的方法中，先在类的定义中对成员函数作引用说明,即只给出函数的原型,然后在类外定义。

类对象（类实例）和类的关系就像整型变量和整型 int 的关系一样。类对象必须先说明后使用。类对象的说明形式如下：

存储类 类的类型名 对象名

存储类为 auto,register,extern 和 static 其中之一，默认值为 auto。

类对象的成员可用直接成员运算符“。”和间接成员运算符“->”访问。

构造函数和析构函数都是类的成员函数，它们可由系统隐含说明，也可由用户定义。构造函数与类同名，在定义类对象时，自动调用它，为对象分配内存，进行必要的初始化和类型转换工作。析构函数在类名前加一个波浪号“～”，它不带任何参数，当局部对象超出作用域时，析构函数被隐含地调用，释放对象占用的内存。 例如：

```
class A{
        int a;
        public:
        A(int i){a=i}      /*定义构造函数*/
        ～A( );             /*定义析构函数*/
```

```
        void mf( );          /*成员函数,引用说明*/
        friend void f(A);  /*友元函数,引用说明*/
}
void A::mf( )               /*类外定义成员函数*/
{
        cout<<a;
}
void f(A obj)               /*类外定义友元函数*/
{
        cout << obj。a;
}
…
A obj(1);                   /*定义 A 的一个对象 obj,调用构造函数*/
obj。mf ( );                /*调用成员函数*/
f(ibj);                     /*调用友元函数*/
```

2.1.3 语言所提供的控制结构

程序语言中控制结构为把数据和数据上的运算组合成程序提供了基本框架，理论上证明了可计算问题的程序都有可用顺序、选择和重复这三种控制结构来描述。

（1）顺序结构：顺序结构用来表示一个计算机操作序列。从操作序列的第一个计算开始，顺序执行序列的计算操作，直至序列的最后一个计算操作。顺序结构内也可以包含其他控制结构。以只有 A、B 两个操作步骤为例，图 2。2 表示顺序结构的流程图。

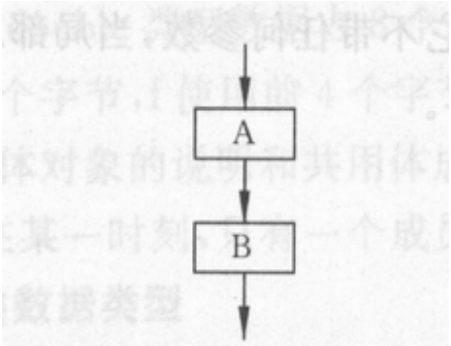


图 2.2 顺序控制结构

（2）条件选择结构：条件选择结构有一条件（P）和两个供选择的操作 A 和 B 组成。先计算条件 P 的值，如果 P 的值为真，则执行操作 A；否则执行操作 B。见图 2。3。当条件选择结构中的 A 或 B 又是由条件选择结构组成时，就呈现嵌套条件选择结构形式。

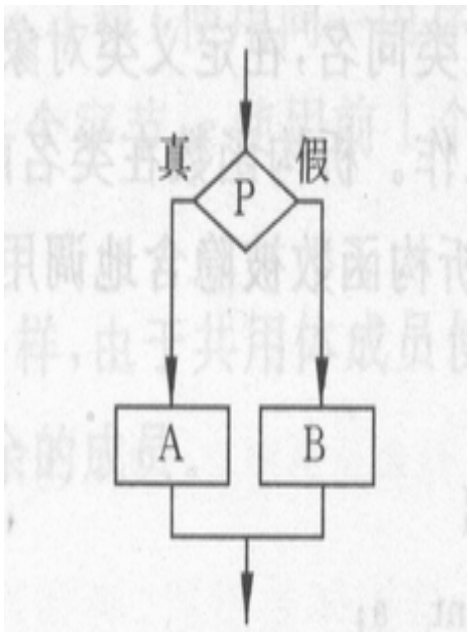


图 2.3 条件选择结构

（3）重复结构：重复结构为程序描述重复计算提供控制手段。重复结构有多种形式，最常见的有以下两种形式：

① while 型重复结构（图 2。4）。当条件 P 成立（真）时，反复执行操作 A；直到 P 不成立（假）时，结束重复操作。

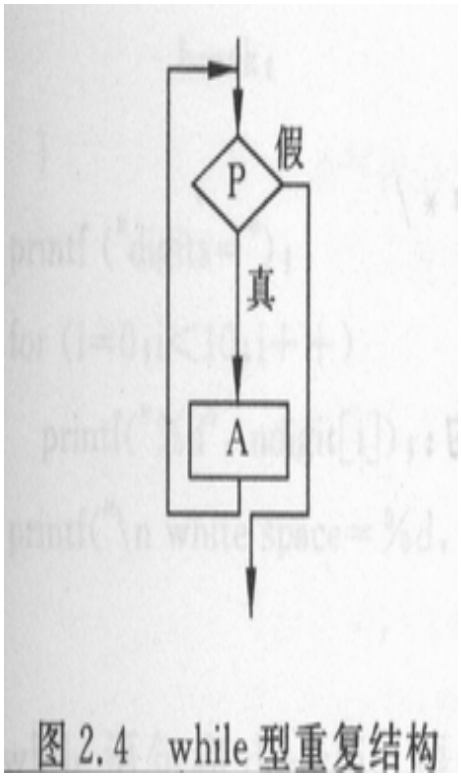


图 2.4 while 型重复结构

② do-while 型重复结构（图 2。5）。先执行操作 A，再求条件 P 的值，若 P 的值为真，再执行操作 A；如此反复，直到 P 的值为假，结束重复。

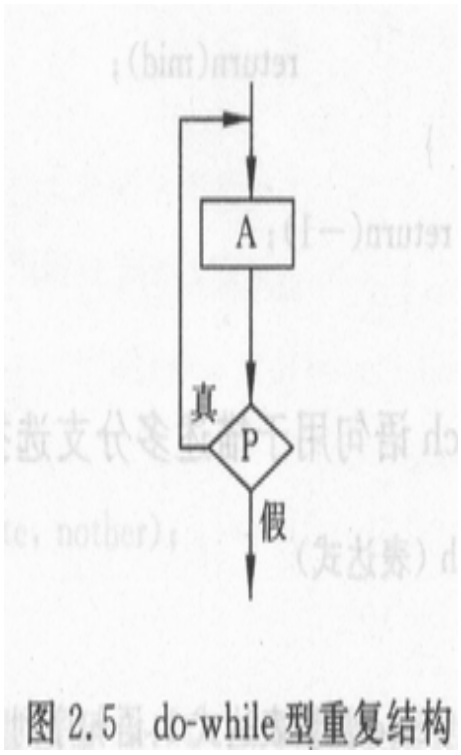


图 2.5 do-while 型重复结构

下面考察 C 和 C++语言所提供的控制结构语句。

（1）复合语句：用于描述顺序控制结构。它将一列语句序列变成一个顺序执行的整体，在逻辑上它是一个单一语句，能用作其他控制结构的成分语句。用花括号将一语句序列括直来就构成了复合语句。

（2）if 语句和 switch 语句：它们都有用于描述条件选择控制结构。if 语句一形式为：

```
if（表达式）    /*当表达式的值为非零时，执行语句 1；否则，执行语句 2*/
    语句 1
else
    语句 2
```

当 else 后的语句 2 为空语句时，可简写成：

```
if（表达式）
    语句
```

注意：因 if 语句的分去语句也可以是 if 语句，即嵌套的 if 语句，C/C++规定 else 总是与它前面最接近的 if 对应。

例如，为了说明三路判断，这里举一个判断一个特定值 x 是否出现在已排好序的数组 v 中的折半查找函数的例子。v 中的元素必须按递增顺序排列。如果 x 出现在数组 v 中,则该函数就返回那个位置(一个 0 到 n-1 的数字);否则就返回(-1)。

```
int binary (int x,int v[],int n )   /*在 v[0]…v[n-1]中找 x*/
{ int low ,high,mid;
  low=0;
  high=n-1;
  while(low<=high) {
      mid=(low+high)/2;
      if (x<v[mid])
            high=mid-1;
      else if (x>v[mid])
```

```
            low=mid +1;
      else                          /*在 v 中找到 x*/
            return(mid);
    }
   return(-1);                      /*x 不在 v 中*/
}
```

switch 语句用于描述多分支选择情况，一般形式为：

```
switch（表达式）
{
        case 常量表达式 1：语句序列 1
                    ……
        case 常量表达式 n ：语句序列 n
        default:语句序列 n+1
}
```

它和 break 语句配合使用。swithch 语句可嵌套的 if 语句来描述，但 switch 语句更清晰地反映了程序的计算过程。

例如，编写一个程序对每一个数字字符、空白字符（空格符、制表符、换行符）以及其他字符的出现次数进行计数。

```
#include < stdio。h>
main()
{ int c,I,nwhite,nother,ndigit[10];
/* ndigit 、nwhite、 nother 分别用于计录数字、空白符号其他字符出现的次数*/
＃include<stdio・h>
main（）
```

```
{int c,I,nwhite,nother,ndigit[10]} ;
 /*ndigit、nwhite、nother 分别用于计录数字、空白符和其他字符出现的次
 数*/
 nwhite=nother=0;
 for(i=0; i<10; i++)
     ndigit[i]=0;
 while((c=getchar())!=EOF)
     switch(c) {
     case′0′:case′1′:case′2′:case′3′:case′4′:
     case′5′:case′6′:case′7′:case′8′:case′9′:
           ndigit[c-′0′]++;
           break;
     case′ ′:case′\n′:case′\t′:
           nwhite++;
           bresk;
           default:
                   nother++;
                   break;
      }
     printf(″digits=″);
     for (i=0; i<10; i++)
       printf(″%d″,ndigit[i]);
     printf(″\n whitespace=%d,other=%d\n″,nwhite,nother);
}
```

(3)while 语句和 do-while 语句:while 语句用来描述 while 型重复控制结构,其一般形式为:

```
while(表达式)
        语句
```

执行 while 语句时,先计算表达式的值,当值为非 0 时,执行内嵌语句；然后重新计算表达式的值,若值为非 0,则再次执行内嵌语句。仅当表达式的值为 0 时,才结束 while 语句的执行。

例如,函数 strcpy(s,t)是把字符串*t 拷贝到字符串*s 中。

```
void strcpy(char * s,char *t)
{
      while(*s++=*t++)
              ;
}
```

do-while 语句用来描述 do-while 型重复控制结构,其一般形式为:

```
do
    语句
while(表达式);
```

执行 do-while 语句时,先执行内嵌语句,然后计算表达式的值,若值为非 0,则重新执行内嵌语句。如此反复,直到表达式的值等于 0 才结束 do-while 语句的执行。

例如,函数 itoa(n,s)是把一个整数 n 转换为一个字符串 s。

```
#define abs (x)((x)<0?-(x):(x))
void itoa (int n,char s[])
{int i,sign;
  void reverse(char s[]);          /*函数 reverse 说明*/
  sign=n;                          /*记录符号*/
```

```
  i=0;
  do{                                  /*反序方式处理产生数字*/
      s[i++]=abs(n%10)+' 0'            /*获取下一数字*/
  }while((n/=10)!=0);                  /*删除*/
  if (sign<0)
      s[i++]= ' —' ;
  s[i]= ' \0 ' ;
  reverse (s) ;
}
/*函数 reverse (s) 是把字符串 s 中和各元素的位置颠倒过来。*/
#include<strihg·h>
void reverse  (char s[ ])
{int c, i, j;
  for (i=0, j=strlen (s) -1; i<j; i++, j--) {
        c=s[i];
        s[i]=s[j];
        s[j]=c;
  }
}
```

（4）for 语句:for 语句是 C 和 C++语言中最为灵活、使用广泛的重复控制结构，其一般形式为：

for（表达式 1；表达式 2；表达式 3）

语句

它的语义为：

```
  表达式 1;
  while（表达式 2）{
          语句
          表达式 3;
  }
```

在 for 语句一般形式中，表达式 1 的作用是为重复控制的有关变量赋初值，表达式 2 用于重复控制条件，表达式 3 用于修正有关变量，语句是重复控制操作。

例如，编写一个 shell 排序的函数。

```
vold shellsort（int v[ ], int n）
{/*按人小到大顺序排序 v[0] …v[n-1]*/
  int gap, i, j, temp;
  for（gap=n/2; gap>0; gap/=2）   /*控制被比较元素间的距离*/
      for（i=gap; i<n; i++）      /*比较指定距离的各对元素*/
            for（j=i-gap; j>=0&&v[j]>v[j+gap]; j-=gap{
                  /*交换未按规定次序排列的元素*/
                    temp=v[j];
                    v[j]=v[j+gap];
                    v[j+gap]=temp;
            }
}
```

2.2 汇编程序基本原理

2.2.1 汇编语言

汇编语言是为特定的计算机或计算机系统设计的面向机器的语言。汇编语言中的语句可分成两大类：一类是与机器指令相对应的可执行汇编语句；另一类汇编语句称为汇编控制语句（亦称伪指令）。用汇编语言编写程序，才能被计算机理解执行。经过汇编程序的工作，可执行汇编语句被转化成对应的机器指令；而伪指令并不翻译成机器指令，它们的作用是控制汇编程序工作。伪指令主要用来告诉汇编程序做一些除了翻译机器指令外必须做的工作。通常汇编语言都应设立下面一伪指令：

（1）定义常数语句：IBM360/370 的汇编语言中定义常数语句的格式是：

名字 DC x，y…

其中，DC 是语句的记忆码，x，y 等规定了用户定义的常数。汇编程序应把所定义常数按规定依次装入以名字为开始单元地址的一系列内存单元中。

虚拟机 COMET 上的汇编语言 CASL 也设立了定义常数伪指令 DS。

（2）定义存储语句：IBM360/370 的汇编语言中用 DS 语句来定义内存单元，例如：TAB1 DS 100C

规定为用户分配 100 个字节内存区域。CASL 语言也设立了定主存储单元的伪指令 DS。

（3）开始语句：一般开始语句用来指出源程序段的开始；IBM360/370 汇编语言中伪指令 START 就是这样的语句。CASL 语言也有 START 语句。

（4）结束语句：结束语句用来指出源程序的结束；IBM360/370 汇编语言和 CASL 语言都设立 END 伪指令表示源程序结束。

每一条汇编语句被划分成四个区，领奖是标号区、操作码区、操作数区和注解区；各个区域之间用确定的符号分隔开。对于标号区中的标号，程序员用它来指示一条汇编语句，它实际上代表内存单元地址。操作码区是该语句操作记忆码，它可以是机器指令记忆码、伪指令记忆码等。操作数区指出本条汇编语句所涉及使用的寄存器操作数或内存操作数等。

2.2.2 汇编程序

汇编程序的功能是将汇语言所编写的源程序翻译成由机器指令和其他信息组成的目标程序。汇编程序的基本工作包括：

（1）将每一条可执行汇编语句转换成对应的机器指令；

（2）处理源程序中出现的伪指令。

这一工作通常不可能一次完成。由于形成操作数地址的各个部分有可能出现符号，而符号会是稍后语句的标号：

SUB 1，C50

…

C50 DC 50

为了计算各汇编语句中的地址，我们在汇编程序中设立单元地址计数器 LC，其初值一般为 0。以后每处理完一条可执行汇编语句和与存储分配有关的伪指令（如定义常数语句、定义存储语句），LC 的值就增加相应长度，这样 LC 的值始终是一个存储单元的相对地址。当处理一条汇编语句的标号时，就将 LC 当时值定义为标号值。由于符号使用可能出现在符号定义前，整个汇编程序工作要通过对源程序进行两次扫描方能完成。

第一次扫描的主要工作是定义符号的值。除了设置单元计数 LC 外，我们设立机器指令表 MOT1。由于本次扫描并不具体生成机器指令，MOT1 表的每一个元素只需两个域：机器指令记忆码和指令长度。在扫描过程中，我们将符号及其值记录在符号表 ST 中。此外，在第一次扫描中，还需要对与定义符号值有关的伪指令进行处理。为了叙述方便，不妨设立伪指令表 POT1，POT1 表的每一个元素只有两个域：伪指点令记忆码和相应处理子程序入口。下面是对汇编程序第一扫描的描述：

1.单元计数器 LC 置初值 0。

2.打开源程序文件。

3.反复执行：

3.1 从源程序文件读下一条语句。

3.2 如果该语句有标号，则将标号和单元计数器 LC 当时值送符号表 ST，

3.3 根据语句操作码，执行：

3.3.1 如果是可执行汇编语句，K 是查 MOT1 表所得机器指令长度，则 LC：LC=LC+K。

3.3.2 如果是伪指令记忆码，则调用 POT1 表相应元素所规定的子程序。

3.3.3 如果是非法记忆码，则调用出错子程序。

直到语句操作码是 END 为止。

4.关闭源程序文件

第二次扫描的目的是产生目标程序。除了前一次扫描所生成的符号表 ST 外，我们需要建立机器指令表 MOT2，该表的元素包含下面区域：机器指令记忆码、机器指令的二进制操作码（binary-code），格式指导（type）和长度（length）。我们还设立第二次扫描适用的伪指令表 POT2，它的每一元素仍是两个域：伪指令记忆码和相应的子处理程序入口；所不同的是，在第二次扫描中，伪指令有着完全不同的处理。

在第二次扫描中，可执行汇编语句应被翻译成对应的二进制代码机器指令。这一工作涉及两个方面；把机器指令记忆码转换成二进制机器指令操作码，以及求出操作数区各操作数的值（用二进制表示）。在些基础上，可以装配出二进制代码的机器指令。对于第一部分工作，只要根据机器指令记忆码查机器指令表 MOT2 就可以获得相应的二进制表示的机器指令操作码。从求值角度看，第二部分工作并不复杂。由于形成操作数地址的各个部分都可以表达式形式出现，统一定义一个过程 eval-expr（index，value）。调用时，只

要将表达式在汇编语句缓冲区 S 的开始位置通过 index 传送给此过程，该过程就通过 value 返回此表达式的值。例如，虚拟计算机 COMET 的机器指令可归属于“X”型指令，其汇编语句为：

OP R1，N2 X2

OP R1，N2

我们可以写出下面处理“X”型指令的程序段（假定 index 已指向操作数在缓冲区 S 的首址）：

```
eval-expr（index，R1）；

index：=index+1；

eval-expr(index,N2)；

if S[index]= ′ , ′ then

      begin

              index：=index+1；

              eval-expr（index，X2）；

      end

else

      X2：=0；
```

其他类型指令的处理操作数的程序段可以类似写出。设当前可执行汇编语句的操作记忆码在 MOT2 表的索引值为 i，则整个可执行汇编语句处理可以描述如下：

```
OP：=MOT2[i]，binary-code；

TYPE：=MOT2[i]，type；

case TYPE of

    ′ X′ ：求 X 型指令操作数各个部分值，然后按规定字节形成指令；

end；
```

将形成指令送往输出区；

在第二次扫描中，伪指令就根据伪指令记忆码，调用 POT2 表相应元素所规定的子程序。DS 伪指令的主要目的是留存储空间。不妨设一个工作单元 K，用以累计以字节为单位的存储空间大小，初值为 0。从 DS 伪

指令的操作区求出 K 的在大小后，就向输出区送 K 个空格以达到保留所规定存储单元的目的。DC 伪指令处理只不过向输出区送的是所转换得到的常量。最后，START 伪指令工作可能输出目标程序开始的标准信息，而 END 伪指令则可能是输出目标程序结束的标准信息，这些信息都是为装配程序提供的。

2.3 解释程序基本原理

解释程序是一种语言处理程序，它直接执行源程序或源程序的内部形式。因此，解释程序并不产生目标程序，这是它和编译程序的主要区别。图 2.6 显示了高级语言实现的四种可能情况。

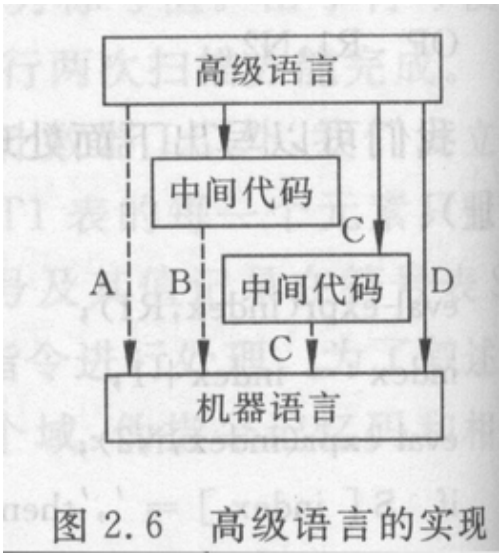


图 2.6 高级语言的实现

在类型 A 的实现方案中，源程序被直接解释执行。这种解释程序对源程序进行逐个字符检查，然后执行规定的动作。例如，如果扫描到字符串序列：

```
GOTO L
```

解释程序就开始搜索源程序中标号 L 后面紧跟冒号“：”的出现位置。这类系统在实现时需要反复扫描程序，这使得按类型 A 方案的解释程序效率很低。

在类型 B 的实现方案中，先将源程序翻译成高级中间代码，然后再扫描高级中间代码，对高级中间代码进行解释执行。通常，在高级中间代码和高级语言代码的语句间存在着一对应关系。APL 和 SNOBOL4 的实现很多都采用这种方法。

类型 C 又是一种解释程序的实现方案，类型 C 的解释程序和类型 B 的解释程序不同点在于，类型 C 的解释程序首先将源程序转化成和机器代码十分接近的低级中间代码，然后再解释执行这种低级中间代码。通常，在这种实现方案下，高级语言的语句和低级中间代码间存在着 1-n 对应关系。PASCAL-P 解释系统是这类解释程序的一个实例，它在词法分析、句法分析和语义基础上，先将源程序翻译成 P-代码；一个非常简单的解释程序解释执行这种 P-代码。这类系统具有良好的可移植性。

类型 D 就是普通的编译程序。在编译程序方案下，高级语言编写的源程序被最终翻译成机器语言来表示的目标程序。这类系统的目标程序执行效率最高。

一般来说，建立在翻译基础上的系统在执行速度上都优于建立在解释执行基础上的系统。翻译系统的缺点是其复杂性，这使得它的开发和维护费用都大。相反，解释系统比较简单，可移植性好，适合于以交互形式执行程序，其缺点是执行速度慢。

下面对解释系统的结构作一扼要的描述。这类系统通常可以分成两部分：第一部分包括通常的词法分析程序以及语法和语义分析程序，它的作用仍是把源程序翻译成中间代码，中间代码的设计常常采用逆波兰表示形式。第二部分是解释部分，用来对第一部分所产生的中间代码进行解释执行，由于真正的解释工作在解释部分完成，下面的介绍仅涉及第二部分。

用数组 MEM 来模拟计算机内存，（源程序的）中间代码和解释部分的各个子程序都存放在数组 MEM 中。全局变量 PC 是一个程序计数器，它记录了当前正在执行的中间代码位置。这种解释部分的常见的总体结构可以由下面两部分组成：

• PC：=PC+1；

• 执行位于 opcode-table[MEM[PC]]的子程序（解释子程序执行后返回前面）。用一个简单例子来说明其工作情况。设两个实型变量 A 和 B，A 与 B 相加的中间代码是：

```
start:  Ipush
            A
        Ipush
            B
        Iaddreal
```

其中，中间代码 Ipush 和 Iaddreal 实际上都是 opcode-table 表的索引值（即位移），而该表单元中存放的值：

```
opcode-table[Ipush]=push
opcode-table[Iaddreal]=addreal
```

就是对应的解释子程序的起始地址，A 和 B 都是 MEM 中的索引值，解释部分开始执行时，PC 的值为 start-1。解释部分可表示如下：

```
interpreter-loop:
        PC：=PC+1;
        goto opcode-table[MEM[PC]];
Push：PC：=PC+1;
```

```
        stackreal(MEM[MEM[PC]]);

        goto interpreter-loop;

addreal:

        stackreal(popreal()+popreal());

        goto interpreter-loop;

…(其余各解释子程序)
```

其中，stackreal（）表示把相应值压入下推栈，而 popreal（）表示把下推栈的栈顶值取出，然后弹出栈顶元素。这儿的下推栈是解释部分工作时的一个工作栈。

2.4 编译程序基本原理

2.4.1 编译过程概述

编译程序的功能是把某高级语言书写的源程序翻译成与之等价的低级语言（汇编语言或机器语言）的目标程序.它是一个复杂的整体的过程.图 2.7 给出了编译过程的各个阶段，这是一种典型的划分方法.事实上，某些阶段可能组合在一起.这些阶段间的源程序的中间表示形式就没有必要构造出来了.

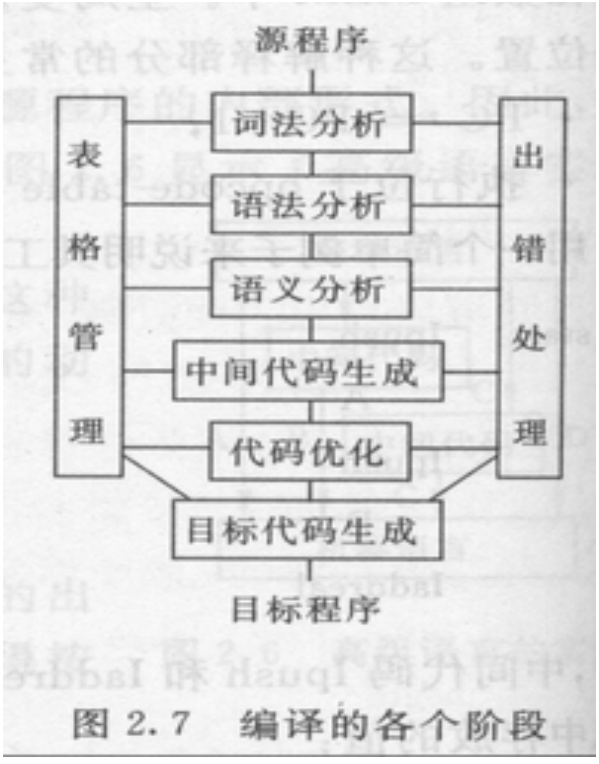


图 2.7 编译的各个阶段

（1）词法分析阶段：是编译过程的第一个阶段.这个阶段的任务是从左到右一个字符地读入程序，识别出一个个单词符号.单词符号是程序设计语言的基本语法符号，如：基本字（也称关键字或保留字）、标识符、常数、运算符和界符.词法分析程序所输出的单词符号常采用以下二元式表示： （单词种别，单词自身的值）

单词的种别是语法分析需要的信息，而单词自身的值则是编译其他阶段需要的信息.

词法分析所依据的是语言的词法规则，即描述单词结构的规则.例如某源程序片断如下：

bgin var snm ，frist，count：real；sum：=frist+count *10 end.

词法分析阶段将构成这段程序的字符组成如下单词序列：

（1）保留字 begin （2）保留字 var （3）标识符 sum （4）逗号，（5）标识符 first

（6）逗号， （7）标识符 count （8）冒号 ：（9）保留字 real （10）分号；

（11）标识符 sum （12）赋值号：= （13）标识符 frist （14）加号+ （15）标识符 count

（16）乘号* （17）整数 10 （18）保留字 end （19）界符 •

经过词法分析后，赋值语句 sum：=first+count*10 则表示为 id1：=id2+id3*10；其中，id1，id2 和 id3 分别表示 sum， first 和 count 三个标识符的内部形式.

(2)语法分析阶段：语法分析的任务是在词法分析的基础上将单词符号序列分解成各类语法单位，如“程序”、“语句”、“表达式”等.语法分析所依据的是语言的语法规则，即描述程序结构的规则.通过语法分析确定整个输入串是否构成一个语法上正确的程序.

上述的 id1：=id2+id3*10 经语法分析得知其是 PASCAL 语言的“赋值语句”，可表示成图 2.8 所示的语法树.

词法分析和语法分析本质上都是对源程序结构进行分析.

（3）语义分析阶段：审查源程序有无语义错误，为代码生成阶段收集类型信息.比如语义分析的一个工作是进行类型审查，审查每个运算符是否具有语言规范允许的运算对象.比如在语句 sum：=first+count*10 中，*两个运算对象：count 是实型，10 是整形，则语义分析阶段进行类型审查之后，在语法分析所得的分析树上增加一语义处理结点，表示整型变成实型的一目运算符 inttoreal，则图 2.8 变成图 2.9.

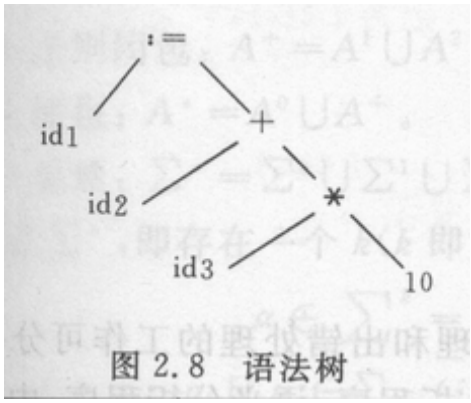


图 2.8 语法树

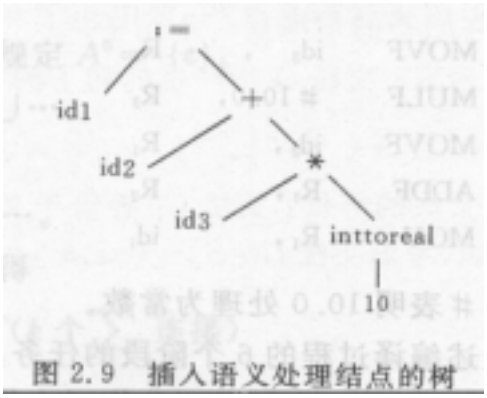


图 2.9 插入语义处理结点的树

（4）中间代码生成阶段：在进行了上述的语法分析和语义分析阶段工作之后，有的编译程序将源程序变成一种内部表示形式，这种内部表示形式叫做中间语言或中间代码.所谓“中间代码”是一种简单、含义明确的记号系统.其设计的原则有两点：一是容易生成，二是容易将它翻译成目标代码.许多编译程序采用了一种近似“三地址指令”的“四元式”作为中间代码，这种四元式的形式为：（运算符.远处对象 1，运算对象 2，结果）。

（5）比如源程序 sum：=first+count*10 可生成四元式序列：

①（inttoreal 10 - t_1）

②（* id_3 t_1 t_2）

③（+ id_2 t_2 t_3）

④（：= t_3 - id_1）

基中 t_i（i=1，2，3）是编译程序生成的临时变量，用于存放运算结果.

语义分析和中间代码生成所依据的是语言的语义规则.

（6）代码优化阶段：此阶段是对前阶段产生的中代码进行变换或进行改造，目的是生成的目标代码更为高级，即省时间和省空间.优化所依据的原则是程序的等价变换规则.比如上述的中间代码可转化成下面的等价代码：

（* id_3 10.0 t_1 ）

（+ id_2 t_1 id_1）

因为编译程序的这个阶段已把将 10 转换成实型数的代码化简掉了，同时 t_3 仅仅用来将其值传递给 id_1 ，也可以被化简掉，这只是优化工作的两个方面、此外优化工作还涉及公共子表达的提取、循环优化等.

（6）目标代码生成阶段：这一阶段的任务是把中间代码变换在特定机器上的绝对指令代码或可重定位的指令代码或汇编指令代码.这是编译的最后阶段，它的工作与硬件系统的结构和指令的含义有关.例如，使用两个寄存器 R_1 和 R_2 可生成上述优化了的四元式的某汇编代码.

①MOVF id_3 , R_2

②MULF #10.0, R_2

③MOVF id_2, R_1

④ADDF R_1, R_2

⑤MOV R_1, id_1

这里有#表明 10.0 处理为常数.

上述编译过程的 6 个阶段的任务，再加上表格管理和出错处理的工作可分别由几个模块或程序完成，它们分别称作词法分析程序、语法分析程序、语义分析程序、中间代码生成程序、代码优化程序、目标代码生成程序、表格管理程序和出错处理程序.从而可给出一个典型的编译程序结构框图.

编译过程中源程序的各种信息被保留在种种不同的表格里，编译各阶段工作都涉及到构造、查找或更新有关的表格.因此，编译程序中必含有一组管理各种表格的程序.

如果源程序有错误，编译程序应设法发现错误，把有关信息报告给用户.这部分工作是由专门的一组出错处理程序完成的，它与编译各阶段都有联系.

2.4.2 文法和语言的形式描述

1.基本定义

（1）字母表 Σ：是元素的非空有穷集合.例如 Σ={α，b}.

（2）字符：字母表 Σ 中的一个元素.例如 a 或 b.

（3）字（字符串）：是 Σ 中的字符所组成的任何有穷序列.例如 a，ab，aaa 都是 Σ 上的字.

（4）字的长度：字中的字符个数.|aba|=3.

（5）空字（空字符串）ε：由零个字符组成的序列，长度|ε|=0.

（6）Σ*：是指 Σ 上所有字的全体；空字 ε 也包括在内.例如 Σ*={ε，a，b，aa，bb，ab，ba，aaa，…}.

（7）字运算：设字 $a\in\Sigma^*$，$\beta\in\Sigma^*$.

（8）连接：字 α，β 的连接 αβ 把字 β 写在 α 之后.显然有 εα=αε=α.

（9）方幂：把字 α 自身连接 n 次得到的字，称为字 α 的 n 次幂，记为 α^n.

$\alpha^0=\varepsilon$，$\alpha^n=\alpha\alpha^{n-1}=\alpha^{n-1}\alpha$（n>0）.

（10）字集运算：设 A，$B\subseteq\Sigma^*$，即 A，B 是字母表 Σ 上的字集.

（11）或（合并）：$A\cup B=\{\alpha\mid\alpha\in A$ 或 $\alpha\in B\}$.

（12）积（连接）：$AB=\{\alpha\beta\mid\alpha\in A$ 且 $\beta\in B\}$.

（13）幂：$A^n=A\cdot A^{n-1}=A^{n-1}A$（n＞0），并规定 $A^0=\{\varepsilon\}$.

（14）正则闭包：$A^+=A^1\cup A^2\cup A^3\cup\cdots\cup A^n\cup\cdots$.

（15）闭包：$A^*=A^0\cup A^+$.

显然：$\Sigma^*=\Sigma^0\cup\Sigma^1\cup\Sigma^2\cup\cdots\cup\Sigma^n\cup\cdots$.

若 $\alpha\in\Sigma^*$，即存在一个 k（k 即为 α 的长度）使得

$\alpha\in\Sigma^k=\Sigma\Sigma\cdots\Sigma$（k 个Σ连接）

亦就是 $\alpha=\alpha_1\alpha_2\cdots\alpha_k$ 且 $\alpha_i\in\Sigma$（i=1，2，…，k）.

2.语言的形式描述

（1）文法的定义：描述语言的语法结构的形式规则称为文法.文法 G 是一个四元组，可表示为

G=（V_T，，V_N，S，P）

V_T是一个非空有限集，每个元素称为终结符；V_N是一个非空有限集，每个元素称为非终结符.

$V_T\cap V_N=\varnothing$ 即 V_T和 V_N不含公共元素.用 V 表示 $V_T\cup V_N$，称 V 为文法 G 的字母表.S 是一个非终结符，称为开始符号；它至少要在一条产生式中作为左部出现.P 是一个有限的具有如下形式的产生式（也称重写规则）的集合：

$\alpha\rightarrow\beta$

其中：α∈V^+且至少含有一个非终结符，称为产生式的左部；β∈V^*称为产生式的右部.若干个产生式 α→β_1，α→β_2…，α→β_n的左部相同，可简写为 α→β_1|β_2|…|β_n；称 β_i（1≤i≤n）为 α 的一个候选式.

（2）句子和语言：设文法 G=（V_T，，V_N，S，P），我们引入下面一些概念：

①直接推导与推导：若 α→β∈P，r 和 δ∈V^*，记号 rαδ $\Rightarrow$ rβδ 称为文法 G 中的一个直接推导.称 rαδ 可直接推导出 rβδ，rβδ 是 rαδ 一个直接推导.显然，对 P 中每一个产生式 α→β 有 $\alpha \underset{G}{\Rightarrow} \beta$ 若在文法 G 中存在一个直接推导序列即 $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \alpha_n$（n>0），称 α_0可以推导出 α_n，α_n是 α_0的一个推导，并记为 $\alpha_0 \overset{+}{\underset{G}{\Rightarrow}} \alpha_n$ 我们用记号 $\alpha_0 \overset{*}{\underset{G}{\Rightarrow}} \alpha_n$ 表示 $\alpha_0=\alpha_n$或 $\alpha_0 \overset{+}{\underset{G}{\Rightarrow}} \alpha_n$.

②直接归约和归约（推导的逆过程）：若文法 G 中有一个直接推导 $\alpha_0 \underset{G}{\Rightarrow} \beta$ ，则称 β 可直接归纳约成 α，α 是 β 的一个直接归约.若文法 G 中有一个推导 $r \overset{*}{\underset{G}{\Rightarrow}} \delta$ 则称 δ 可归约成 r，r 是 δ 的一个归约.

③句型和句子：若文法 G 的开始符号为 S 且有如下推导 $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ ， α∈V^*，则称 α 是文法 G 的一个句型，若 X 是文法 G 的一个句型，且 $X \in V_T^*$，则称 X 是文法 G 的一个句子，即仅含终结符的句型是一个句子.

④语言：文法 G 所产生的句子的全体是一个语言，记为 L（G），它可表示成：

$$L(G)=\left\{X \middle| S \overset{*}{\underset{G}{\Rightarrow}} X \text{ 且 } X \in V_T^* \right\}$$

⑤文法的等价：文法 G_1等价于文法 G_2是指 L（G_1）=L（G_2）

例如，考虑文法 $G_1[S]$=（{α，b}，{S，A，B}，S，P）其中 P 为：

S→AB

A→αA|α

B→bB|b

我们可以分析得出 L（G_1）={$a^m b^n$|m，n≥1}.

例如，构造一个文法 G_2 使 $L(G_2)=\{a^nb^n \mid n\geq 1\}$. G_2 和 G_1 的区别在于，G_2 的每个句子中 a 和 b 的个数必须相同.我们可以写出文法 $G_2[S]$：

$$S \rightarrow aSb \mid ab$$

（3）文法的分类：乔姆斯基（chomsky）把文法分成四种类型，即 0 型、1 型、2 型和 3 型.这几类文法的差别在于对产生式施加不同的限制.设 $G=(V_T, V_N, S, P)$，若对它的每条产生式分别加上以下的第 i 条限制，则可得 I 型文法：

①0 型：G 的每一条产生式 $\alpha \rightarrow \beta$ 满足 $\alpha \in V^+$且至少含有一个非终结符，而 $\beta \in V^*$.

②1 型：G 的任何产生式为 $\alpha \rightarrow \beta$ 满足 $|\alpha| \leqslant |\beta|$，仅仅 $S \rightarrow \varepsilon$ 例外，但 S 不得出现在任何产生式右部.

③2 型：G 的任何产生式为 $A \rightarrow \beta$，$A \in V_N$，$\beta \in V^*$

④3 型：G 的任何产生式为 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$，$\alpha \in V_T^*$，$A, B \in V_N$.

0 型文法也称短语文法，其能力相当于图灵机.1 型文法也称上下文有关文法，其能力相当于线性界限自动机.2 型文法也称不上下文有关文法，其能力相当于非确定的下推自动机.3 型文法也称右线性文法（或正规文法），其能力相当于有限自动机.

3.上下文无关文法

它是 2 型文法，现今的程序基本上可用它来描述.

（1）规范推导（最右推导）：如果在推导的任何一步 $\alpha \Rightarrow \beta$ 其中 α，β 是句型，都是对 α 中的最右（最左）非终结符进行替换，则称这种推导为最右（最左）推导。最右推导常为规范推导.

（2）短语、直接短语和句柄：设 $\alpha\ \beta\ \delta$ 是文法 G 的一个句型 ，即 $S \overset{+}{\Longrightarrow} \alpha\ \beta\ \delta$，且满足 $S \overset{+}{\Longrightarrow} \alpha A \delta$，$A \overset{+}{\Longrightarrow} \beta$ 则称 β 是句型 $\alpha\ \beta\ \delta$ 相对于非终结符 A 的短语.特别，如有 $A \Rightarrow \beta$ 则称是句型 $\alpha\ \beta\ \delta$ 相对于产生式 $A \rightarrow \beta$ 的直接短语.一个句型最左直接短语称为该句型的句柄.

（3）素短语：素短语是一个短语，它至少含有一个终结符，除它本身外不再含有更小的素短语.

例如定义表达式文法： $G_1[E]=(\{+, *, (,), i\}, \{E, T, F\}, E, P)$

其中 P 为：

$$E \rightarrow T \mid E+T$$

$$T \rightarrow F \mid T*F$$

$$F \rightarrow (E) \mid i$$

考虑文法 $G_1[E]$的一个句型 i*i+i.为叙述方便，我们将句型写作 $i_1*i_2+i_3$.因为有：

$E \Rightarrow^{*} F^{*}i_2+i_3$，且 $F \Rightarrow i_1$，则称 i_1是句型 $i_1*i_2+i_3$的相对于非终结符 F 的短语，也是相对于产生式 F→i 的直接短语.又有：

$E \Rightarrow^{*} i_1*+F+i_3$，且 $F \Rightarrow i_2$，则称 i_2是句型 $i_1*i_2+i_3$的相对于 F 的短语，也是相对于产生式 F→i 的直接短语.还有：

$E \Rightarrow^{*} i_1*i_2+F$，且 $F=i_3$，则称 i_3也是句型 $i_1*i_2+i_3$的相对于 F 的短语，也是相对于产生式 F→i 的直接短语.还有：

$E \Rightarrow^{*} T*i_2+i_3$，且 $T \Rightarrow^{+} i_1$，则 i_1是句型 $i_1*i_2+i_3$的相对于 T 的短语.还有：

$E \Rightarrow^{*} T+i_3$，且 $T \Rightarrow^{*} i_1*i_2$，则 i_1*i_2是句型 $i_1*i_2+i_3$的相对于 T 的短语.还有：

$E \Rightarrow^{*} i_1*i_2+T$，且 $T \Rightarrow^{+} i_3$，则 i_3是句型 $i_1*i_2+i_3$的相对于 T 的短语.还有：

$E \Rightarrow^{*} E+i_3$，且 $E \Rightarrow^{+} i_1*i_2$，则 i_1*i_2是句型 $i_1*i_2+i_3$的相对于 T 的短语.还有：

$E \Rightarrow^{*} E$，且 $E \Rightarrow i_1*i_2+i_3$，则 $i_1*i_2+i_3$是句型 $i_1*i_2+i_3$的相对于 E 的短语.

即 i_1，i_2，i_3，i_1*i_2和 $i_1*i_2+i_3$都是句型 $i_1*i_2+i_3$的短语，而且 i_1，i_2，i_3均是直接短语，其中 i_1是最左直接短语，即句柄.

虽然 i_2+i_3是句型 $i_1*i_2+i_3$的一部分，并不是它的短语，因为尽管有 $E \Rightarrow^{+} i_2+i_3$，但不存在从文法开始符 E 到 i_1*E 的推导.

又如上述文法 G_1［E］的一个句型 F*F+i，F*F、i 和 F*F+i 都是它的短语，而 F*F 和 i 是它的素短语，F*F 也是它的最左素短语.

(4)规范归约：设 α 是文法 G 的一个句子，若序列 α_n，α_{n-1}…，α_1，α_0满足：

- $\alpha_0=\alpha$；
- α_0为文法的开始符号，即 $\alpha_0=S$；

·对任何 i(0＜i≤n)， α_{i-1}是 α_i经把句柄替换成相应产生式的左部符号而得到的.规范归约是最左归约，它是最右推导的逆过程.

(5)语法树：我们可以用一张图表示一个句型的推导，这种表示称为语法树(也称推导树).

给定文法 G=(V_T，V_N，S，P)，对于 G 的任何句型都能构造与之关联的语法树，这棵树满足下列 5 个条件：

①每个结点都有一个标记，此标记是 V(U{ε})的一个符号；

②根的标记 S；

③若一结点 n 是内部结点，并且有标记 A，则 A 肯定在 V_N中；

④如果结点 n 是直接子孙，从左到右的次序是结点 n_1，n_2，…，n_k，其标记分别为 A_1，A_2，…，A_k，那么 $A\rightarrow A_1A_2\cdots A_k$一定是 P 中的一个产生式；

⑤若结点 n 的标记为 ε ，那么结点 n 是叶子，且是它父亲的惟一儿子.

(6)文法的二义性：如果一个文法存在某个句子对应两棵不同的语法树，则说这个文法是二义的.

2.4.3 词法分析

词法分析的任务是把构成源程序的字符串序列转换成单词符号串的中间程序.词法规则可用三型文法(正规文法)或正规表达式描述，它产生的集合是 Σ^*上的一个子集，我们称为正规集.转换方法有人工的状态转换图方法和有限自动机的自动方法.

1. 正规表达式

对于字母表 Σ 而言，正规式和它所表示的正规集的递归定义为：

(1) ε 和∅是正规式，它们所表示的正规集分别为{ε}和∅；

(2)任何 α∈Σ， α 是 Σ 上的一个正规式，它所表示的正规集为{α}；

(3)假定 U 和 V 都是 Σ 上的正规式，它们所表示的正规集分别为 L(U)和 L(V)，那么，(U|V)，(U·V)和$(U)^*$也都是正规式，它们所表示的正规集分别为 L(U)∪L(V)，L(U)L(V)(连接积)和$(L(U))^*$(闭包).

(4)仅由有限次使用上述三步骤定义表达式才是 Σ 上的正规式，仅由这些正规式所表示的字集才是 Σ 上的正规集.

其中的"|"读为"或"，"·"读为"连接"，"*"读为"闭包"(即任意有限次的自重复连接).规定算符的优先顺序为：先"*"，次"·"，最后"|".连接符"·"可省略不写.

例如，令 Σ={a，b}， Σ 上的正规式和相应的正规集的例子有：

正规式	正规集
ba^*	Σ 上所有以 b 为首后跟任意多个 α 的字
$a(a\|b)^*$	Σ 上所有以 α 为首的字
$(a\|b)^*(aa\|bb)(a\|b)^*$	Σ 上所有含有两个相继的 α 或两个相继的 b 的字.

若两个正规式所表示的正规集相同，则认为二者等价.两个等价的正规式 U 和 V 记为 U=V.例如，$b(ab)^*=(ba)^*b$，$(a|b)^*=(a^*b^*)^*$.设 U，V 和 W 均为正规式，正规式服从的代数规律有：

①U|V=V|U（交换律）

②U|(V|W)=(U|V)|W（结合律）

③U(VW)=(UV)W（结合律）

④U(V|W)=UV|UW（分配律）

(V|W)U=VU|WU

⑤ εU=Uε=U

2. 有限自动机

有限自动机作为一种识别装置，它能准确地识别正规集.有限自动机分为两类：确定的有限自动机和不确定的有限自动机.

(1)确定的有限自动机(DFA)：一个确定的有限自动机(DFA)M 是一个五元组：

$M=(S, \Sigma, f, S_0, Z)$

其中：

①S 是一个有限集，它的每个元素称为一个状态；

②Σ 是一个有穷字母表，它的每个元素称为一个输入字符；

③f 是转换函数，是在 $S\times\Sigma\rightarrow S$ 上的单值部分映照.如 $f(s, a)=s'$（$s, s'\in S, a\in\Sigma$）就意味着：当前状态为 s，输入字符为 a 时，将转换到下一状态 s'，我们称 s'为 s 的一个后继状态；

④$s_0\in S$，是惟一的一个初态；

⑤$Z\subseteq S$，是一个终态集(可空).

一个 DFA 可以表示成一个状态图(或称状态转换图)，也可以用一个状态转换矩阵表示.

例如，DFA M=({S，U，V，Q}，{a，b}，f，s，{Q})，其中f为：

f (S，a)=U f (V，a)=U

f (S，b)=V f (V，b)=Q

f (U，a)=Q f (Q，a)=Q

f (U，b)=V f (Q，b)=Q

所对应的状态图为图 2.10，状态转换矩阵为图 2.11.　理.上述分析过程也可看成自底向上构造语法树的过程.

LR 分析法是一种规范归约分析法，它根据当前分析栈中的符号串(通常以状态表示)和向右顺序查看输入串的 k 个(k≥0)符号，就可惟一确定分析器的动作是移进还是归约，以及用哪条产生式归约，因而也就能惟一地确定句柄.当 k=1 时，已能满足当前绝大多数高级语言编译程序的需要.常用的 LR 分析器有 LR(0)，SLR(1)，LALR(1)和 LR(1)四种.

一个 LR 分析器由三个部分组成.

①总空程序：也称驱动程序.对所有 LR 分析器总控制程序都是相同的.

②分析表：不同的文法具有不同的分析表.同一文法采用不同的 LR 分析器时，分析表也不同.分析表又可分为动作表(ACTION)和状态转换表(GOTO)两个部分，它们都可用二维数组表示.

③分析栈：包括文法符号栈和相应的状态栈，它们均是先进后出栈.

分析器的动作由栈顶状态和当前输入符号所决定(LR(0)分析器不需向前查看输入符号).LR 分析器工作过程示意图如图 2.17 所示.

其中 SP 为栈顶指针，S［i］为状态栈，X［i］为文法符号栈.状态栈转换表用 GOTO［S_i，X］=S_j表示，规定当栈顶状态为 S_i遇到当前文法符号(终结符或非终结符)时应转向状态 S_j.ACTION［S_i，a］规定了栈顶状态 S_i时遇到输入符号 a 应执行的动作.动作有四种可能：

①移进：把 S_j=GOTO［S_j，a］移进状态栈，a 移进文法符号栈.其中 i，j 表示状态号.

②归约：当在栈顶形成句柄 β 时，则把 β 归约为相应产生式 A→β 的非终结符 A.若 β 的长度为 r(即｜β｜=r)，则 SP 减去 r，并把 A 移入文法符号栈内，S_j=GOTO［S_j，A］移进状态栈，其中 S_i为修改 SP 后的栈顶状态.

③接受 acc：当归约到文法符号栈中只剩文法的开始符号 S，并且输入符号串已结束时，即当前输入符是"#"，分析成功.

④报错：当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，报错，说明输入串不是该文法的能接受的句子.

LR 分析器的关键部分是分析表的构造，在此不作介绍.

(2)算符优先分析法：算符优先分析法主要是定义文法中终结符之间的优先关系，利用这种关系定义和操作可归约串，从而达到语法分析的目的.基本概念定义如下：

①算符文法：文法事不存在形如 P→…QR…的产生式，其中 P，Q 和 R 均为非终结符，则称 G 为算符文法.

②终结符间的优先关系：对一个不含 ε 产生式的算符文法 G，任何一对终对符 a 和 b 具有如下优先关系：

• a $\doteq$ b 当且仅当 G 中含有形如 P→…ab…或 P→…aQb…的产生式；

• a＜• b 当且仅当 G 中含有形如 P→…aR…的产生式，且 R $\overset{+}{\Rightarrow}$ b…或 R $\overset{+}{\Rightarrow}$ Qb…；

• a • ＞b 当且仅当 G 中含有形如 P→…Rb…的产生式，且 R $\overset{+}{\Rightarrow}$ …a 或 R $\overset{+}{\Rightarrow}$ …aQ；

③算符优先文法：若文法 G 是一个不含 ε 产生式的算符文法，且对任意两个终结符(a，b)至多只有＜• 、• ＞和 $\doteq$ 三种关系中的一种成立，则称 G 是一个算符优先文法.

④算符优先关系表的构造：首先定义两个集合：FIRSTVT(P)，LASTVT(P).

• FIRSTVT(P)={a|P $\overset{+}{\Rightarrow}$ a 或 P $\overset{+}{\Rightarrow}$ Qa…，a∈V_T，P，Q∈V_N}

求法：

a.若有产生式 P→a…或 P→Qa…，则 a 加入 FIRSTVT(P)；

b.若有产生式 P→R…且 b∈FIRSTVT(R)，则 b 加入 FIRSTVT(P)；

c.重复 b，直至每一个 FIRSTVT 集合不再扩大为止.

• LASTVT(P)={a|P $\overset{+}{\Rightarrow}$ …a 或 P $\overset{+}{\Rightarrow}$ …aQ，a∈V_T，P，Q∈V_N}

a.若有产生式 P→…a 或 P→…aQ，则 a 加入 LASTVT(P)；

b.若有产生式 P→…Q 且 b∈LASTVT(Q)，则 b 加入 LASTVT(P)；

c.重复 b，直至每一个 LASTVT 集合不再扩大为止.

有了文法中的每个非终结符 P 的 FIRSTVT(P)和 LASTVT(P)的值，可用下面的算法构造文法的优先关系表.

FOR 每条产生式 P→X_1，X_2…X_n DO
  FOR i：=1 TO n-1 DO
    BEGIN
      IF X_i和X_{i+1}均为终结符，THEN 置 X_i 〓 X_{i+1}；
      IF i≤n-2 且 X_i和X_{i+2}均为终结符，但 X_{i+1}为非终结符.
      THEN 置 X_i 〓 X_{i+2}
      IF X_i为终结符而 X_{i+1}为非终结符.
      THEN FOR FIRSTVT(X_{i+1})中的每个 b DO 置 X_i＜・b；
      IF X_i为非终结符而 X_{i+1}为终结符
      THEN FOR LASTVT(X_i)中的每个 a DO 置 a・＞X_{i+1}
    END

由上述算法可自动构造算符文法 G 的算符优先关系表，从而可判断 G 是否为算符优先文法.例如，考虑下面文法 G［E'］：

(0)E'→#E#

(1)E→E+T|T

(2)T→T*F|F

(3)F→P↑F|P

(4)P→(E)|i

我们可以求出每个非终结符的 FIRSTVT 集和 LASTVT 集；

	FIRSTVT	LASTVT
E'	#	#
E	+ * ↑ (i	+ * ↑) i
T	* ↑ (i	* ↑) i
F	↑ (i	↑) i
P	(i	) i

文法 G 的算符优先关系表为：

	+	*	↑	i	(	)	#
+	⋗	⋖	⋖	⋖	⋖	⋗	⋗
*	⋗	⋗	⋖	⋖	⋖	⋗	⋗
↑	⋗	⋗	⋖	⋖	⋖	⋗	⋗
i	⋗	⋗	⋗			⋗	⋗
(	⋖	⋖	⋖	⋖	⋖	≐	
)	⋗	⋗	⋗			⋗	⋗
#	⋖	⋖	⋖	⋖	⋖		≐

⑤算符优先分析算法：算符优先分析用"最左素短语"来刻画可归约串.算符优先文法中的任一句型(括在两个#之间)为$\#N_1a_1N_2a_2\cdots N_na_nN_{n+1}\#$，其中，第个 a_i 都是终结符，N_i 是可有可无的非终结符.句型中的最左素短语是满足如下条件的最左子串 $N_ja_jN_{j+1}a_{j+1}\cdots N_ia_iN_{i+1}$：

$a_{j-1}\lessdot \cdot a_j \doteq a_{j+1} \doteq \cdots \doteq a_i \cdot \gtrdot a_{i+1}$

下面给出分析算法，其中 S 是符号栈，用以寄存归约或待形成最左素短语的符号串，工作单元 a 存放当前读入的终结符号，k 代表栈 S 的使用深度.

```
K：=1：S［k］：='#'；
REPEAT
    把下一个输入符号读进 a 中；
    IF S［k］∈VT THEN j：=k ELSE j：=k-1：

    WHILE S[j] ·>a DO

        BEGIN

        REPEAT

          Q：=S[j]；

          IF S[j-1] ∈VT THEN j：=j-1 ELSE j：j-2
        UNTIL S［j］·<·Q；
```

```
            把 S［j+1］…S［k］归约为某个 N；
            k：=j+1；
            S［k］：=N
      END OF WHILE；
      IF S［j］＜·a OR S［j］ a THEN
            BEGIN k：=k+1：S［k］：=a END
      ELSE ERROR  /*调用出错诊察程序*/
UNTIL a=`#'
```

在正确的情况下，算法工作完毕时，栈 S 中只剩#N#.

2.自顶向下分析方法

自顶向下分析法也称面向目标的分析方法，也就是从文法的开始符号出发企图推导出与输入的单词串相匹配的句子，若输入串是给定文法的句子，则必能推出，反之必然出错.自顶向下分析法又可分为确定的和不确定的两种，确定的分析需对文法有一定的限制，但由于实现方法简单、直观，便于手工构造或自动生成语法分析器.不确定的方法即带回溯的分析方法这种方法实际上是一种穷举的试探方法，效率低，代价高，因而极少使用.

确定的自顶向下分析方法不允许文法含有任何的左递归，并要找出克服回溯的条件，从而可根据当前输入符惟一地确定选用哪个产生式替换相应非终结符往下推导，或构造一棵相应的语法树.

通过消除文法的左递归和提取产生式的公共左因子，有一类被称为 LL(1)文法可用递归子程序方法或预测分析法实现确定的自顶向下的语法分析.

(1)消除左递归：一个文法是左递归时不能采用自顶向下分析法.

①消除直接左递归：一般而言，假定关于 P 的全部产生式是：

$P \rightarrow Pa_1|Pa_2|\cdots|Pa_m|\beta_1|\beta_2|\cdots|\beta_n$

其中，$a_i(1\leqslant i\leqslant m)$都不等于 ，$\beta_i(1\leqslant i\leqslant n)$都不以 P 开头，那么，消除 P 的直接左递归后改写为：

$P\rightarrow \beta_1P'|\beta_2P'|\cdots|\beta_nP'$

$P'\rightarrow a_1P'|a_2P'|\cdots|a_mP'|\varepsilon$

②消除文法中一切左递归的算法：要求文法中不含回路即无 $P \overset{+}{\Rightarrow} P$ 的推导，也不含以 ε 为右部的产生式.

算法步骤：

(i)把文法的所有非终结符按某一顺序排序，如 P_1，P_1，…，P_n.

(ii)

```
FOR i：=1 TO n DO
  BEGIN
      FOR j：=1 TO i-1 DO
          BEGIN
            若 Pj的所有产生式为： Pj→ δ1| δ2|…| δk
            替换形如 Pi→Pjr 的产生式变为：Pi→ δ1r| δ2r|…| δkr
          END
      消除 Pi 中的一切直接左递归
  END
```

(iii)最后删除无用产生式，即去除那些从文法开始符出发永远无法到达的非终结符的产生式.

例如，文法 G［S］：

S→Qc|c

S→Rb|b

S→Sa|a

虽不具有直接左递归，但 S、Q、R 都有左递归的，例如有：

$S \Rightarrow Qc \Rightarrow Rbc \Rightarrow Sabc$

我们要消除它的左递归.令它的非终结符的排序为 R、Q、S.对于 R，不存在直接左递归.把 R 代入到 Q 的有关候选式后，我们把 Q 的规则变为

Q→Sab|ab|b

现在的 Q 同样不含直接左递归，把它代入到 S 的有关候选式后，S 变成

S→Sabc|abc|bc|c

经消除了 S 的直接左递归后，我们得到整个文法为：

S→abcS'|bcS'|cS'

S'→abcS'|ε

Q→Sab|ab|b

R→Sa|a

显然，其中关于 Q 和 R 的规则已是多余的.经化简后所得文法是：

S→abcS'|bcS'|cS'

S'→abcS'|ε

注意，由于对非终结符排序的不同，最后所得的文法在形式上可能不一样，但它们都是等价的.例如，若对文法 G 的非终结符排序选为 S、Q、R，那么，最后所得的无左递归文法是：

S→Qc|c

Q→Rb|b

R→bcaR'|caR'|aR'

R'→bca|R'|ε

(2)提取公共左因子：一个文法中的产生式含有公共左因子时，不能采用不带回溯的自顶向下的分析法.

假定关于 A 的产生式为：$A\rightarrow \alpha\beta_1|\alpha\beta_2|\cdots|\alpha\beta_n$

提取公共左因子后改写为：

A→aA'

$A\rightarrow \beta_1|\beta_2|\cdots|\beta_n$

经反复提取，就能使每个非终结符(包括新引进的)候选式的首符集两两不相交，其代价是增加产生式和 ε 产生式.

(3)LL(1)文法：我们首先定义两个集合：FIRST(α)、FOLLOW(A).

①$FIRST(\alpha)=\{\alpha\mid \alpha \overset{*}{\Rightarrow} \alpha\cdots,\ \alpha\in V_T,\ \alpha\in V^*\}$

特别是，若 $\alpha \overset{*}{\Rightarrow} \varepsilon$，则规定 $\varepsilon\in FIRST(\alpha)$.

求法：(i)对每一文法符号 X∈V 计算 FIRST(X).

a.若 $X\in V_T$，则 X 加入 FIRST(X).

b.若 $X\in V_N$，考察它的产生式：

若 $X\rightarrow \alpha\cdots,\ \alpha\in V_T$，则 α 加入 FIRST(X)；

若 $X\rightarrow \varepsilon$，则 ε 加入 FIRST(X)；

若 $X \to R_1, R_2 \cdots, R_k$，且 $R_1 \in V_N$，则先将 FIRST(R_1)/ε (所有非 ε 元素)加入 FIRST(X)，然后在 $R_1, R_2 \cdots\cdots R_{i-1} \overset{*}{\Rightarrow}$ ε (意味着：对每个 j：$1 \leqslant j \leqslant i-1$， $\varepsilon \in$FIRST(R_j))时，将 FIRST(R_i)/ε 加入 FIRST(X).特别是，若 $\varepsilon \in$FIRST(R_j)， j=1，2，…，k，将 ε 加入 FIRST(X).

c.反复执行 b 步直到每个符号的 FIRST 集不再增大为止.

(ii)对 $\alpha \in V^*$，计算 FIRST(α)，设 $\alpha = X_1 X_2 \cdots X_n$，$X_i \in V$，$1 \leqslant i \leqslant n$.

a.置 FIRST(α)=FIRST(X_1)/ε .

b.若 $\varepsilon \in$FIRST(X_j)， j=1，2，…，i-1，$2 \leqslant i \leqslant n$，则将 FIRST($X_i$)/ε 加入到 FIRST(α)中；特别是，若 ε FIRST(X_j)， j=1，2，…，n，则 ε 加入到 FIRST(α)中.

②FOLLOW(A)={a|$S \overset{*}{\Rightarrow} \cdots Aa \cdots$，$a \in V_T$，$a \in V_N$}

其中，S 是文法开始符.特别是 $S \overset{*}{\Rightarrow} \cdots A$，则规定#∈FOLLOW(A).

求法：对每一个 $A \in V_N$，逐个考察产生式：

(i)将#加入 FOLLOW(S)，S 为文法开始符.

(ii)若 $A \to \alpha B \beta$ 是一条产生式，$B \in V_N$，则把 FIRST(β)/ε 加入 FOLLOW(B).

(iii)若 $A \to \alpha B$ 是一条产生式或 $A \to \alpha B \beta$ 且 $\beta \overset{*}{\Rightarrow} \varepsilon$ 即($\varepsilon \in$FIRST(β))，则将 FOLLOW(A)加入 FOLLOW(B).

(iv)反复使用(ii)、(iii)直到每个非终结符的 FOLLOW 集不再增大为止.

一个上下文无关文法 G 是 LL(1)文法的充分必要条件是：对每一个非终结符 A 的任何两个不同的产生式 $A \to \alpha$，$A \to \beta$，满足：

(i)FIRST(α)∩ FIRST(β)=$\varnothing$，其中 α， β 不同时能 $\overset{*}{\Rightarrow} \varepsilon$.

(ii)如果 $\beta \overset{*}{\Rightarrow} \varepsilon$，那么 FIRST(α)∩ FOLLOW(A)=$\varnothing$.

某些非 LL(1)文法能通过消除文法中的左递归，提取公共左因子的文法等价变换，改写成 LL(1)文法.

(4)递归子程序分析法：是比较简单、易于构造的一种事法分析方法，要求文法满足 LL(1)文法.它的实现思想是对文法中每个非终结符编写一个递归过程，每个过程的功能是识别由该非终结符推出的串，当某非终结符的产生式有多个候选式时能够按 LL(1)形式惟一地确定选择某个候选式进行推导.

(5)预测分析方法：它是自顶向下的另一种方法，一个预测分析器由三个部分组成：预测分析程序、先进后出栈和预测分析表.预测分析表与文法有关，可用一个二维数组 M 表示，其元素 M[A，a](A∈V_N，a∈V_TU{#})存放一条关于 A 的产生式，表明当用 A 向下推导时，面临输入符会 a 时，所应采用的候选式；当元素内容无产生式时，则表明用 A 向下推导时，遇到了不该出现的符号，因此转向出错处理.其分析表的构造如下：

①对文法 G 的每条产生式 A→a 执行(ii)、(iii)两步；

②对每个 a∈FIRST(a)，把 A→a 加至 M［A，a］中；

③若 ε ∈FIRST(a)，则对每个 b∈FOLLOW(A)，把 A→a 加至 M［A，b］中；

④把所有无定义的 M［A，a］标上"出错标志".

显然，若 G 的分析表 M 不含多重定义入口，则 G 是一个 LL(1)文法.此分析表也称 LL(1)分析表.

预测分析程序的工作过程是：在分析开始时，先把"#"和文法开始符依次放入栈中；分析的任何时候都是根据当前栈顶符号 X 和当前的输入符 α 行事：

若 X=a='#'，则分析成功，若 X='#'，而 X≠a，则出错；

若 X∈V_T且 X=a，则 X 退栈，读下一字符入 a，若 X∈V_T，而 X≠a，则出错；

若 X∈V_N，且 M［X，a］中为 X→a，则 X 退栈，a 反序进栈(对 X→ ε 无符号进栈)；若 M［X，a］为"出错标志"，则调用出错诊断程序 ERROR.

例如，构造文法 G［E］：

E→TE'

E'→+TE'｜ε

T→FT'

T'→*FT'｜ε

F→(E)|i

的 LL(1)分析表.我们先给出有关的 FIRST 集和 FOLLOW 集：

	FIRST	FOLLOW
E	(i	#)
E'	+ ε	#)
T	(i	#) +
T'	* ε	#) +
F	(i	#) + *

文法 G 的 LL(1)分析表为：

	i	+	*	(	)	#
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F′→i			F→(E)		

2.4.5 语法制导翻译和中间代码生成

在语法分析过程中，随着分析的步步进展，一条产生式获得匹配(自顶向下分析)或用于归约(自底向上分析)时，根据这条产生式所对应的语义子程序进行翻译的方法称作语法制导翻译.它既适用于自顶向下分析，又适用于自底向上分析，在此只介绍自底向上分析的制导翻译.

1. 中间代码的形式

编译程序使用的中间代码有多种形式，常见的有逆波兰、三元式、四元式和树形表示.
例如：e：=(a+b)*(c+d)的各种表示形式如下：

逆波兰(也称后缀式)表示：e ab+cd+*：=

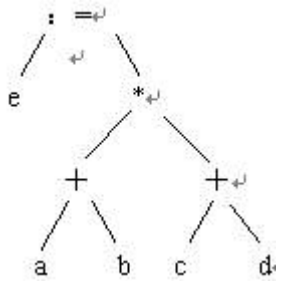
三元式表示：①(+，a，b)

②(+，c，d)

③(*，(1)，(2))

④(：=，(3)，e)

树表示：



四元式表示：①(+a，b，t_1)

②(+，c，d，t_2)

③(*，t_1，t_2，t_3)

④(：=，t_3，-，e)

下面将用四元式表示语句，以及翻译各种语句的语义描述.

2.几种语法单位的翻译

对于各种语法单位的翻译，要按如下几步进行：①语法单位的文法描述；②语法单位的语义；③根据语义，设计语法单位翻译后的目标结构；④根据目标结构，改写文法并设计相应的语义子程序.首先引进一些翻译时要用到的语义变量和语义过程：

Entry(i)是对标识符 i 查找符号表以获知它在表中的位置(入口)的函数过程.

E.Place 是和非终结符 E 相联系的语义变量，表示存放 E 值的变量名在符号表的入口或整数码(若此变量是临时变量).

GEN(OP，ARG1，ARG2，RESULT)是一个语义过程，它把四元式(OP，ARG1，ARG2，RESULT)填进四元式表中.

NXQ 指示器，指向下一个将要形成但尚未形成的四元式地址(编号).NXQ 的初值为 1，每当执行一次 GEN 后，NXQ 将自动累增 1.

Merg(P_1，P_2)是把 P_1 和 P_2 为链首的两条链合并为一条链，并返回合并后的链首值的函数.

Backpatch(p，t)是把 p 所链接的每条四元式的第四区段都填为 t 的过程.

(1)控制语句中布尔表达式的翻译

①文法 E→E∧E｜E∨E｜┐E｜(E)｜i｜i rop i

其中，rop 是关系符(如＜、≤、=、≠、＞或≥)，运算符的优先级和结合律遵照通常的习惯.

②语义：作为程序中语句执行次序的控制条件.

③目标结构：设计成一系列转移指令.语义变量 E. TC 和 E. FC 分别表示 E 的"真"、"假"出口转移目标.一般而言，E→E(1)∧ E(2)，E→E(1)∨E(2)目标结构如图 2.18 所示.

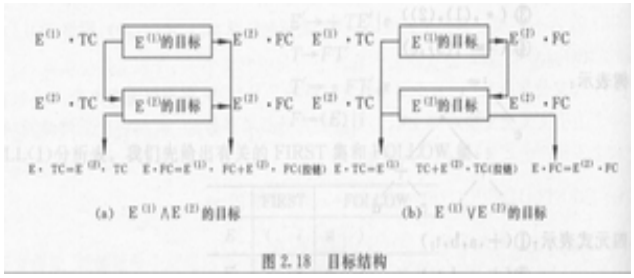


图 2.18 目标结构

④改写文法和相应的语义子程序.

E→E(1)∧E(2)， 为记住 E(2)的首条四元式的地址，需改为：

E^→E(1)∧ { Backpatch(E(1)·TC，NXQ)； |*填真出口*|

E^·FC：=E(1)·FC |*传假出口*|}

E→E^E(2) {E·TC：= E(2)·TC； |*传真出口*|

E·FC：=Merg(E^·FC，E(2)·FC)|*合并假出口*|}

E→E(1)∨E(2) 同样改写为：

E∨→E(1)∨ { Backpatch(E(1)·FC，NXQ)； |*填假出口*|

E∨·TC：=E(1)·TC|*传真出口*| }

E→E∨E(2) { E·TC：=Merg(E∨·TC，E(2)·TC)： |*合并真出口*|}

E·FC：= E(2)·FC： | *传假出口*|}

E→┐E(1) { E·TC：= E(1)·FC ； E·FC：= E(1)·TC | *交换真假出口*|}

E→(E(1)) { E·TC：= E(1)·TC ； E·FC：= E(1)·FC |*传真假出口*|}

E→i { E·TC：=NXQ； E·FC：=NXQ+1；

GEN(jnz，Entry(i)，-，0)；

GEN(j，-，-，0)}

E→i(1) rop i(2) { E·TC：=NXQ；E·FC：=NXQ+1；

GEN(jrop，Entry(i(1))， Entry(i(2))，0)；

GEN(j, -, -, 0)}

(2)条件语句的翻译

①文法：S→if E then $S^{(1)}$| if E then $S^{(1)}$ else $S^{(2)}$

②语义：then 与最近 else 配对，E 为真转向 $S^{(1)}$，否则，转向 $S^{(2)}$.S 执行后转向的位置，由它所处语句环境而定.

③目标结构：语义变量 S.chain 的值是一条链，它把所有那些四元式串在一起，这些四元式期待在翻译 S 之后回填转移目标;真正的回填工作将在处理 S 的外层环境的某一适当时候完成.目标结构如图 2.19 所示.

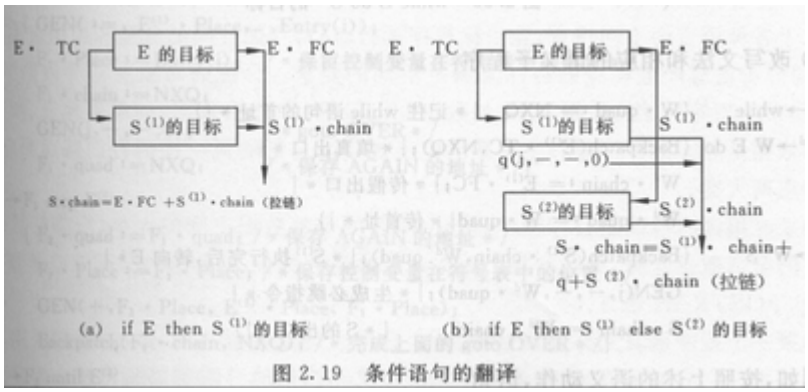


图 2.19 条件语句的翻译

④改写文法与相应的语义子程序.

C→if E then { Backpatch(E·TC, NXQ)： |*填真出口*|

C.chain：= E·FC |*传假出口*|}

S→$CS^{(1)}$ {S. chain：= Merg(C·chain , $S^{(1)}$. chain) |*拉链*|}

T^{p}→$CS^{(1)}$ else {q：=NXQ;

GEN(j, -, -, 0);

T^{p}·chain：= Merg($S^{(1)}$·Chain, q)； |*拉链*|

Backpatch(C.chain , NXQ) |*填假出口*| }

S→$T^{p}S^{(2)}$ { S·chain：= Merg(T^{p}·chain, $S^{(2)}$·chain)|*拉链*| }

(3)循环 while 语句的翻译

①文法：S→while E do $S^{(1)}$

②语义：E 为真执行 $S^{(1)}$，否则，S 执行完.

③目标结构：语义变量 W·quad 用来记住 while 语句所对应的第一条四元式的地址，如图 2.20 所示.

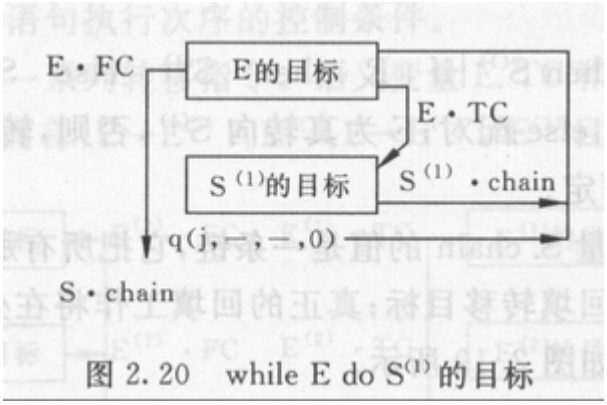


图 2.20 while E do $S^{(1)}$ 的目标

④改写文法和相应的语义子程序

W→while {W.quad：=NXQ |*记住 while 语句的首址*|}

W^d→W E do { Backpatch($E^{(1)}$·TC，NXQ)； |*填真出口*|

W^d·chain：= $E^{(1)}$·FC； |*传假出口*|

W^d·quad：= W·quad |*传首址*|}

S→$W^dS^{(1)}$ { Backpatch($S^{(1)}$.chain， W^d.quad)； |* $S^{(1)}$执行完后，转向 E*|

GEN(j，-，-， W^d.quad)； |*生成必跳指令*|

S.chain；= W^d·chain |*S 的出口*|}

例如，按照上述的语义动作，语句.

while(a＜b)do

if(c＜d)then x：=y+z；

将被翻译成如下的一串四元式：

100(j＜，a，b，102)
101(j，-，-，107)
102(j＜，c，d，104)
103(j，-，-，100)
104(+，y，z，T)
105(：=，T，-，x)

106(j, -, -, 100)
107

(4)循环 for 语句的翻译

①文法 S→for i: = $E^{(1)}$ step $E^{(2)}$ until $E^{(3)}$ do $S^{(1)}$

②我们按 ALGOL 语言的意义来翻译这种循环句.假定 $E^{(2)}$总是正的，在这种假定下，上述循环句的 ALGOL 意义等价于:

i: =$E^{(1)}$
goto OVER
AGAIN: i: =i+$E^{(2)}$
OVER: if i≤$E^{(3)}$ then
begin $S^{(1)}$: goto AGAIN end;

③改写文法和相应的语义子程序

F_1→for i: =$E^{(1)}$

{GEN(: =, $E^{(1)}$·Place, -, Entry(i));

F1·Place: =Entry(i); |*保留控制变量在符号表中的位置*|

F1·chain: =NXQ;

GEN(j, -, -, 0); |* goto AGAIN *|

F1·quad: =NXQ; |*保存 AGAIN 的地址*|}

F_2→F_1 step $E^{(2)}$

{F2·quad: =F_1·quad; |*保存 AGAIN 的地址*|

F_2·place: =F_1·Place; |*保留控制变量在符号表中的位置*|

GEN(+, F_1·Place, $E^{(2)}$·Place, F_1·Place);

Backpatch(F_1·chain, NXQ); |*完成上面的 goto OVER *|}

F_3→F_2 until $E^{(3)}$

{F_3·quad: =F_2·quad;

q: =NXQ;

GEN(j≤, F_2·Place, $E^{(3)}$·Place, q+2)

|*若 i≤$E^{(3)}$转去执行循环体的第一个四元式*|;

F3·chain:=NXQ;

GEN(j, -, -, 0); |*转离循环*|}

S→F_3 do $S^{(1)}$

{GEN(j, -, -, F_3·quad); |*goto AGAIN*|

Backpatch($S^{(1)}$·chain, F_3·quad):

S·chain:=F_3·chain; |*转离循环的转移目标留待处理外层 S 时再回填*|}

例如，循环语句

for i:=1 step 1 until N do M:=M+I;

将被翻译成如下的四元式序列：

100 (:=, "1", -, i)
101 (j, -, -, 103)
102 (+, i, "1", i)
103 (j≤, i, N, 105)
104 (j, -, -, 108)
105 (+, M, i, T)
106 (:=, T, -, M)
107 (i, -, -, 102)
108

(5)动态存储分配和过程调用的翻译

由于各种语言的不同特点，在目标程序运行时，对存储空间的分配和组织有不同的要求，在编译阶段应产生相应的目标来满足不同的要求.分配的对象有初等数据类型(如整、实和布尔型等)、结构数据类型(如数组和记录等)和连接数据(如返回地址、参数等).分配的依据是名字的作用域和生存期的定义规则.分配的策略按语言的特点而定，有静态和动态两大类.

如果在编译时能确定目标程序运行中所需的全部数据空间的大小，安排好目标程序运行时的全部数据空间，并确定每个数据对象的存储位置，则称这种分配策略是静态存储分配，如 FORTRAN 语言.

如果一个程序语言允许递归过程、可变数组或可变数据结构，那么，就需采用动态存储管理技术.它有两种方式：栈式和堆式.栈式动态存储分配策略是将整个程序的数据空间设计为一个栈，适用于 PASCAL，C，ALGOL 之类的语言.每当调用一个过程时，它所需的数据空间就分配在栈顶.每当过程工作结束时，就释放这部分空间.如果一个程序语言提供用户自由地申请数据空间和退还数据空间的机制(C++中的new、delete，PASCAL

的 new)，或者不仅有过程而且有进程的程序结构，即空间的使用未必服从"先申请后释放，后申请先释放"的原则，那以栈式的动态存储分配方案就不适用了，这种情况下通常使用一种称为堆式的动态存储分配方案.下面仅就一个简单的栈式分配，说明过程调用和过程说明的翻译.

考虑一种简单的程序语言结构：没有分程序结构，过程定义不嵌套，但允许过程递归调用.图 2.21 是过程活动记录的内容，SP 总是指向现行过程活动记录的起点，TOP 始终指向(已占用)栈顶单元.

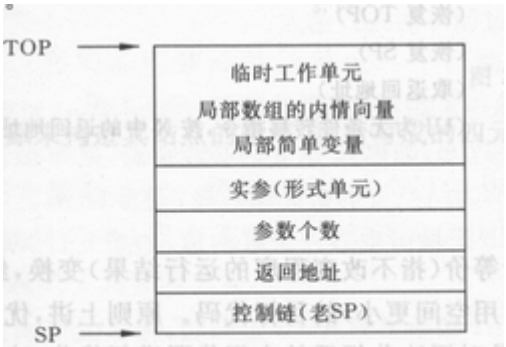


图 2.21 无嵌套定义的过程活动记录

过程调用语句 Call P(T_1，T_2，…，T_n)的四元式序列是：

par T_1
…
par T_n
call p，n

在运行时， par 和 call 应产生如下的目标：

① par T_i的目标：
(i+3)[TOP]：=T_i(传递参数值)
或
(i+3)[TOP]：=addr(T_i)(传递参数地址)

② call P，n 的目标：
1[TOP]：=SP (保护现行 SP)
3[TOP]：=n (传递参数个数)
JSR P (转子指令，转向过程 P 的第一条指令)

转进过程 P 后，就进入过程说明的目标：

• 首先 SP：=TOP+1 (定义新 SP)
1[SP]：=返回地址(保留返回地址)
TOP：=TOP+L (定义新 TOP，L 是过程 P 的活动记录所需单元数，编译时可静态确定)

• 过程说明中数组说明的目标，调整 TOP 的目标；

·过程体内执行语句的目标(用 SP 变址方式访问对象)；

·退出过程语句 return(E)目标：
R：=E·Place (把 E 的值放到特定寄存器 R 中，调用段将从中获得被调用过程的结果值)
TOP：=SP-1 (恢复 TOP)
SP：=0[SP] (恢复 SP)
X：=2[TOP] (取返回地址)
UJ 0[X] (UJ 为无条件转移指令，按 X 中的返回地址实行变址转移)

2.4.6 代码优化

优化是对程序进行等价(指不改变程序的运行结果)变换，经变换后的程序能生成更有效(运行时间更短、占用空间更小)的目标代码.原则上讲，优化可以在编译的各个阶段进行，但最主要的一类是对语法分析后的中间代码进行优化，这类优化不依赖于具体的计算机.另一类重要的优化是在生成目标代码时进行的，它在很大程序上依赖于具体的计算机，在此不作讨论.根据优化所涉及的程序范围，分为局部优化、循环优化和全局优化三个不同的级别.

1.局部优化

局部优化是在基本块(只有一个入口、一个出口的基本程序块)上进行的优化，包括删除多余运算(公共子表达式)、合并已知量和删除无用赋值.

(1)基本块的划分：基本块的入口语句为下列三种情况之一：

·程序的第一条语句；

·条件转移语句或无条件转移语句的转移目标语句；

·紧跟在条件转移语句后面的语句.

求出程序中各个入口语句后，对于每一入口语句，它所属的基本块是由该入口语句到下一入口语句(不句括下一入口语句)或到一转移语句(不包括下一入口语句)或到一转移语句(包括该转移语句)，或到一停语句(包括该停语句)之间的语句序列组成.凡未能纳入某一些基本块的语句都是程序中控制流程无法到达的语句，把它们删除掉.

(2)优化算法：将基本块内的四元式序列转换成带标记或附加信息的无环路有向图 DAG.

例如；构造以下基本块 G 的 DAG(见图 2 .22)

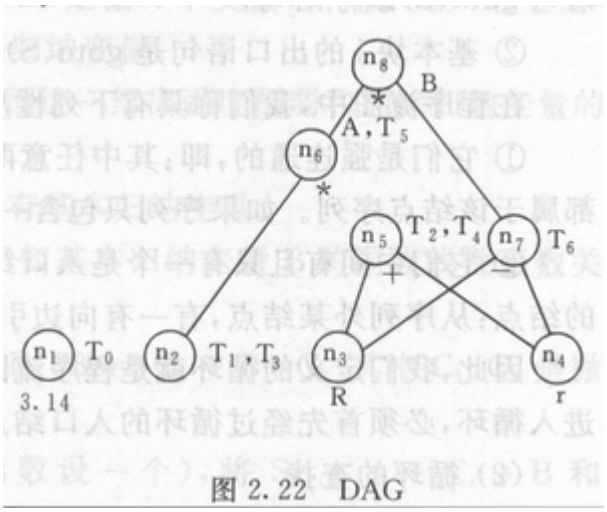


图 2.22 DAG

① T_0：=3.14
② T_1：=2*T_0
③ T_2：=R+r
④ A：=T_1*T_2
⑤ B：=A
⑥ T_3：=2*T_0
⑦ T_4：=R+r
⑧ T_5：=T_3*T_4
⑨ T_6：=R-r
⑩ B：=T_5*T_6

根据上面的 DAG，按原来构造其结点的顺序，重新写成的四元式序列 G’为：

① T_0：=3.14
② T_1：=6.28
③ T_3：=6.28
④ T_2：=R+r
⑤ T_4：=T_2
⑥ A：=6.28*T_2
⑦ T_5：=A
⑧ T_6：=R-r
⑨ B：=A*T_6

把 G’和原基本块 G 相比，可以看出：

• G 中的四元式②和⑥已知量都已合并.

• G 中的四元式⑤的无用赋值已被删除.

• G 中的四元式③和⑦的公共子表达示 R+r 只计算一次，删除了多余运算.

2.控制流程分析和循环优化

(1)控制流程图(科称流图)是具有惟一首结点的有向图.所谓首结点，就是从它开始到控制流程图中任何结点都有一条通路的结点.控制流程图可表示成三元组 G=（N，E，n_0），其中 N 代表图中所有点结点集(结点就是程序的基本块)，E 代表图中所有有向边集，n_0代表首结点(即包含序第一条语句的基本块).E 是这样构成的，假设流图中结点 i 和结点 j 分别对应于程充的基本块 i 和基本块 j，则当下述两条件有一个成立时，从结点 i 有一有向边引向结点 j：

①基本块 j 在程序中紧跟在基本块 i 之后，并且基本块 i 的出口语句不是无条件转移语句 goto(S)或停语句：

②基本块 i 的出口语句是 goto(S)或 if…goto(S)， 并且(S)是基本块 j 的入口语句.

在程序流程图中，我们称具有下列性质的结点序列为一个循环：

①它们是强连通的，即：其中任意两个结点之间必有一条通路，而且该通路上各结点都属于该结点序列.如果序列只包含一个结点，则必有一有向边从该结点引到其自身.

②它们中间有且只有一个是入口结点.所谓入口结点，是指该序列中具有下列性质的结点：从序列外某结 ，有一有向边引到它，或者它就是程序流图的首结点.

因此，我们定义的循环就是程序流图中具有惟一入口结点的强连通子图，从循环外要进入循环，必胯首先经过循环的入口结点.

(2)循环的查找

①必经结点：在程序流图中，对任意两个结点 m 和 n，如果从流图的首结点出发，到达 n 的任一通路都要经过 m，则称 m 是 n 的必经结点，记为 m DOM n.流图中结点 n 的所有必经结点的集合，称为结点 n 必经结点集，记为 D(n).

②回边：假设 a→b 是流程图中的一条有向边，如果 b DOM a，则称 a→b 是流图中的一条回边.

③利用回边求循环：如果已知 n→d 是回边，那么就可求出由它组成的循环.该循环由结点点 n 以及有通路到达 n 而该通路不经过 d 的所有结点组成，并且 d 是该循环的惟一入口结点.

(3)循环化

在找出了程序流图中的循环之后，就可以针对每个循环进行优化.因为循环内的语句是重复执行的，故而循环优化在整个优化工作中是非常重要的.循环优化包括代码外提、强度削弱、删除归纳变量、循环展开和循环合并.实现循环优化，需要在整个程序范围内对所有变量的定值和引用之间的关系进行分析，这就需要先进行数据流分析.在此仅简述三种重要循环优化技术

①代码外提：将循环中的不变运算放到循环外执行.外提步骤：首先在循环的入口结点前增加一个前置结点(新的基本块)，在循环中查找不变运算，按求得的不变运算的次序，把满足外提条件的不变运算提到前置结点中.

②强度削弱：指把程序中执行时间较长的运算替换为执行时间较短的运算.例如把循环中的乘法运算用递归加法运算来替换.

③删除归纳变量：基本归纳变量和归纳变量的定义如下：

·基本归纳变量：如果循环中对变量 I 只能惟一的形如 I：=I ± C 的赋值，且其中 C 为循环不变量，则称 I 为循环中的基本归纳变量.

·归纳变量：如果 I 是循环中一基本归纳变量，J 在循环中的定值总可化归为 I 的同一线性函数，即 J=C_1*I ± C_2 ，其中 C_1 和 C_2 是循环不变量，则称 J 为归纳变量，并称它与 I 同族.一个基本归纳变量也是归纳变量.

删除归纳变量是在强度削弱之后进行的.下面统一给出强度削弱和删除归纳变量的步骤：

(i)利用循环不变运算信息，找出循环中的所有基本归纳变量.

(ii)找出所有其他归纳变量 A，并求出它与已知基本归纳变量 B 的同族线性函数关系 $F_A(B)$.

(iii)(强度削弱)对(ii)找出的归纳变量 A 和 $F_A(B)=C_1*B+C_2$ 其中 C_1，C_2 是循环不变量，执行以下步骤：

(1)设临时变量 $S_{F_A(B)}$(相同的同族线性函数设一个)，将 $S_{F_A(B)}$：=C_1*B 和 $S_{F_A(B)}$：= $S_{F_A(B)}$+C_2(若 C_2=0，此式不要)放到前置结点原有四元式的后面；

(2)循环中原来对 A 的赋值改成 A：= $S_{F_A(B)}$；

(3)在循环中基本归纳变量 B 的惟一赋值 B：=B ± E(E 为循环不变量)后面增加 $S_{F_A(B)}$：= $S_{F_A(B)}$ ± C_1*E，如果 $C_1 \neq 1$ 且 E 是变量名，则此式是以下两个四元式： T：=C_1*E， $S_{F_A(B)}$：= $S_{F_A(B)}$ ± T，其中 T 是临时变量.

(iv)对每一归纳变量 A， 如果在 A：= $S_{F_A(B)}$ 与循环中任何引用 A 的四元式之间没有对 $S_{F_A(B)}$的赋值，且 A 在循环出口之后不活跃，则删除 A：= $S_{F_A(B)}$，并把所有引用 A 的地方改为引用 $S_{F_A(B)}$.

(ｖ)(删除基本归纳变量)如果基本归纳变量 B 在循环出口之后不活跃，并且在循环中，除在其自身的递归赋值中被引用外，只在形如 if B rop Y goto Z 中被引用，则：

(a) 选取一个与 B 同族的归纳变量 M，并设 $F_M(B)=C_1*B+C_2$：

(b) 设临时变量 R，并用 R：=C_1*Y， R：=R+C_2 和 if $S_{F_M(B)}$ rop R goto Z 来替换) if Brop Y goto Z：

(c) 删除循环中对 B 递归赋值的四元式.

例如，考察图 2.23 的流图.

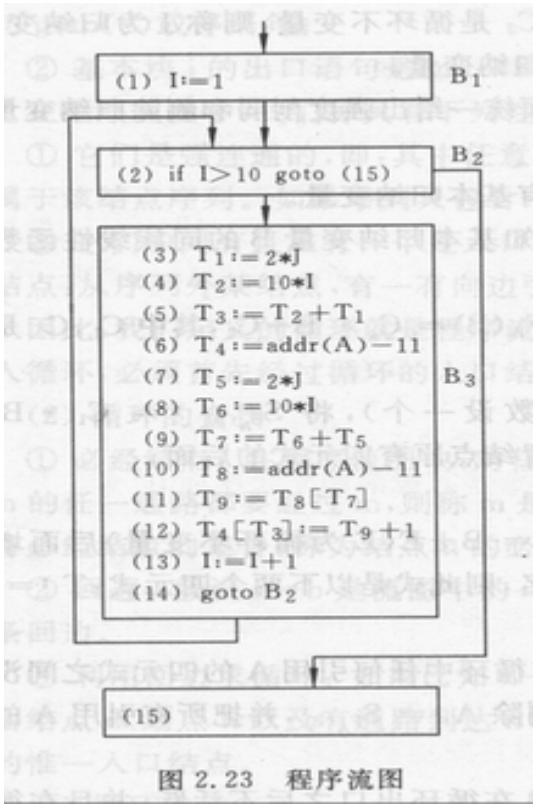


图 2.23 程序流图

容易看出，{B_2，B_3}是循环，B_2是循环入口结点.

代码外提：(3)，(6)，(7)，(10)均可外提到循环的前置结点中，如图 2.24 所示.其中 B_2' 是新建立的循环前置结点.

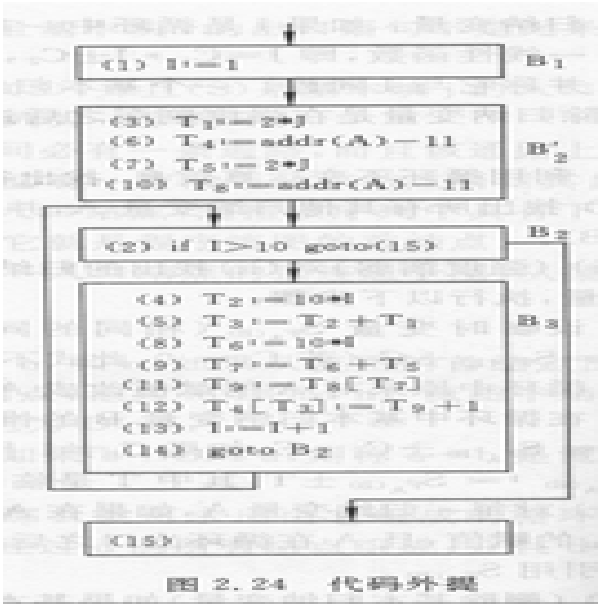


图 2.24 代码外提

整理人：刘海云
整理时间：2003.05

强度削弱：基本归纳变量 I:=I+1 ，归纳变量有 T_2:=10*I， T_6:=10*I， T_3:=T_2+T_1=10*I+T_1， T_7:=T_6+T_5=10*I+T_5.

所以建立 $S_{F_{T_2}}$， $S_{F_{T_6}}$ ，和 $S_{F_{T_7}}$. 强度削弱后的结果见图 2.25 所示.

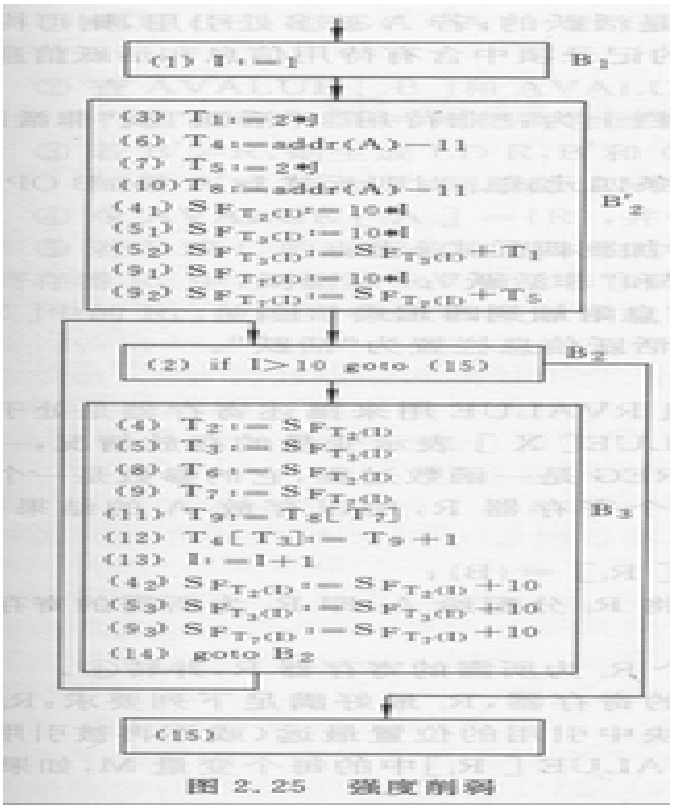


图 2.25 强度削弱

删除归纳变量：删除归纳变量后的结果如图 2.26 所示.

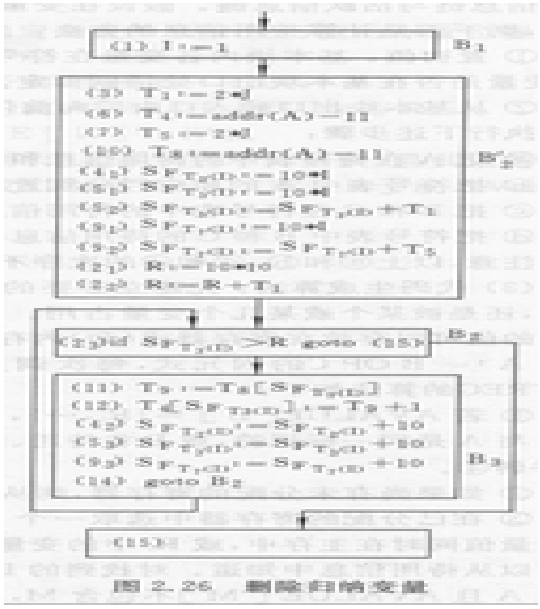


图 2.26 删除归纳变量

3. 全局优化

全局化是在整个程序范围内进行的优化，它包括常数传播、合并已知量、删除全局公共子表达式以及是复写传播(我们称形为 A：=D 的赋值为复写).由于全局优化需先进行数据流分析，在此不作介绍.

2.4.7 代码生成

代码生成是把经过语法分析或优化后的中间代码作为输入，将其转换成特定机器的机器语言或汇编语言代码作为输出，这样的转换程序为代码生成器，代码生成器的设计要着重考虑两个问题：一是如何使生成的目标代码短(空间上)，另一是如何充分利用寄存器和指令的特点(时间上).

1 一个简单的代码生成器

以四元式的中间代码为输入，将其转换成目标代码，并且在一个基本块内考虑如何充分利用寄存器的问题，即每次操作的结果尽可能留在寄器中，不存主存；目标代码中尽可能利用变量在寄存器中的值，不访问主存；在基本块入口时，变量的值在主存，在基本块出口时，若变量的

值在寄存器中，则存入主存；对基本块内不再引用的变量所占用的寄存器及早释放，以提高寄存器的使用效率.

(1)计算机模型：指令形式为 OP R，X，若 OP 是二目运算符，其意义为(R)OP(X)=>R；若是一目运算符，其意义为 OP(X)=>R.其中 R 是寄存器，X 可以是寄存器 R，内存 M，变址或间接方式.

(2)待用信息链：如果在一个基本块中，变量 A 在四元式 i 中被定值，在 i 后面的四元式 j 中要引有 A 值，且从 i 到 j 之间没有其他对 A 的定值点，则称 j 是四元式 i 中对变量 A 的待用信息(即下一个引用点)，同时也称 A 是活跃的；若 A 被多处引用，则可构成待用信息链与活跃信息链.假设在变量的符号表的记录项中含有待用信息和活跃信息的栏目，则下面是计算待用信息的方法：

①置初值：基本块内各变量在符号表中相应栏上为：非待用活跃或非活跃(按变量是否在基本出口处活跃而定).

②从基本块出口到入口由后向前依次处理每条四元式.对四元式 i：A：=BOPC，依次执行下述步骤：

- 把 A 在符号表中的待用信息和活跃信息附加到四元式上；
- 把符号表中 A 的相应栏分别置为非待用和非活跃；
- 把 B 和 C 在符号表中待用信息和活跃信息附加到四元式 i 上：
- 把符号表中 B 和 C 的待用信息栏置为"i"，活跃信息栏置为"活跃" .

注意，以上的次序不能颠倒.

(3)代码生成算法：先建立必要的信息：数组 RVALUE 用来描述寄存器是处于空状态；还是被某个或某几个变量占用.数 AVALUE[X]表示变量的存放情况，一个变量的值可以存放在寄存器或(和)内存中.GETREG 是一函数过程，它的参数是一个形如 i：A：=B OP C 的四元式，每次调用，返回一个寄存器 R，用以存放 A 的结果值.GETREG 的算法如下：

①若 AVALUE[B]={R_i，…}，RVALUE[R_i]={B}：

且 B 与 A 是同一标识符，或 B 非待用，非活跃；则将 R_i 分配给 A，即 R_i 为所需的寄存器 R，并转④.
如果尚有未配的寄存器，则从中选用一个 R_i，为所需的寄存器 R，并转④.

②如果尚有未分配的寄存器，则从中选用一个 Ri 为所需的寄存器 R，并转④.

③在已分配的寄存器中选取一个 R_i，为所需的寄存器，R_i 最好满足下列要求：R_i 中的变量值同时在主存中，或 R_i 中的变量值在基本块中引用的位置最远(或不再被引用)，这可以从待用信息
中知道.对找到的 R_i，对于 RVALUE[R_i]中的每个变量 M，如果 M 不是 A 且 AVALUE[M]不包含 M，则需完成：

(1)生成 ST R_i ，M，即把不是 A 的变量值由 R_i 中送入内存；

(2)如果 M 不是 B，则令 AVALUE[M]={M}，否则，令 AVALUE[M]={M，R_i}.

(3)删除 RVALUE[R_i]中的 M.

(4)给出 R，返回.

这样，一旦得到一条四元式运算的操作寄存器 R，就可进行代码生成，其代码生成算法如下：

对基本块中的每条四元式 i：A：=B OP C 依次进行下述步骤：

①调有 GETREG(i：A：=B OP C)，当返回时，得到一个寄器 R，用作存放 A 的现行值.

②查 AVALUE[B]和 AVALUE[C]，确定变量 B 和 C 现行值的存放位置 B’和 C’(如现行值在寄存器中，则把寄存器取作 B’和 C’）.

③若 B’≠R，则生成 LD R，B’和 OP R，C’，否则，生成 OP R，C’；如果 B’或 C’为 R，则删除 AVALUE[B]或 AVALUE[C]中的 R

④令 AVALUE[A]={R}，并令 RVALUE[R]={A}.

⑤若 B 或 C 的现行值在基本中块中不再被引用，出基本块后不再活跃，且其现行值在某寄存器 R_k 中，则删除 RVALUE[R_k]中的 B 或 C 以及 AVALUE[B]或 AVALUE[C]中的 R_k，使该寄存器不再为 B 或 C 所占用.

第 3 章操作系统基础知识

3.1 操作系统概述

现代计算机系统由硬件和软件两部分构成。硬件指的是构成计算机系统的物理设备，比如控制器、运算器、存储器、设备、通道等；软件是不同于硬件的其他部

分，比如，为方便计算机操作、简化程序设计的内部程序等。操作系统是运行在计算机硬件上最基本的系统软件.操作系统控制和管理所有的系统硬件（如处理器、存储器、各种设备），也控制和管理系统中所有的软件（如系统软件、实用软件、应用软件和用户软件等）。操作系统对计算机使用者提供了一种良好的操作环境，也为各种应用系统提供了基本的支持环境。当代的计算机，无论是超级巨型机，还是中、小型机；也无论是台式个人计算机、便携机，还是连接多台计算机网络，都离不开操作系统。 没有任何软件支持的计算机称为“裸机”，让用户用裸机工作是复杂的、困难的，操作系统在硬件之上建立了一个服务体系，为各种软件提供强有力的支持，呈现在用户面前的计算机是一个方便、友好的环境界面。

现代计算机系统中硬件与软件之间的关系常可分成若干层次，如图 3.1 所示。

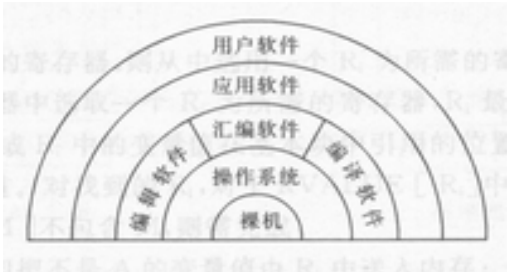


图 3.1 操作系统与硬件软件的关系

由图 3.1 可见，硬件（裸机）在最里层，是计算机系统工作的物质基础，它的外面是操作系统。操作系统对计算机系统中的各类资源（处理器、存储器、设备、数据等)进行管理和提供方便用户使用的多种服务功能,隐蔽对硬件的复杂操作，从而把裸机改造成功能更强、使用更方便的系统。 操作系统是最基本的系统软件。操作系统的外面是其他系统软件,用来对用户的操作和开发软件提供支持(如本文编辑、汇编程序、编译程序、连接程序、命令解释程序等），它们不是操作系统的一部分，一般与操作系统一起由供应商提供。此外，人们也将系统实用软件、系统工具软件等视作系统软件。

系统软件外面是应用软件，它们是为各应用领域服务的软件，如数据库软件、财务软件等。

用户可以直接经系统软件操纵计算机；也可通过实用软件或工具软件操纵计算机，根据用户的需求产生用户软件，如事务处理等；还可直接使用应用软件满足自己的需要。

3.1.1 什么是操作系统

操作系统是一个大型的软件系统,是为了提高计算机系统资源(硬件和软件资源)的利用效率并方便用户使用的一组程序，这些程序可以用软件实现，也可以用固件（微程序设计）实现。

1. 用户观点

用户见到的是如图 3.1 所示的虚拟机（或扩展机）。操作系统是用户与计算机之间的接口，有了操作系统，用户可以方便地作用计算机。操作系统虚拟机在逻辑功能上与裸机不同，即操作系统在裸机上运行，其他软件在操作系统上运行。虚拟机的扩展包括系统功能及数量上的扩展。在功能上，操作系统提供功能很强的系统调用，用户软件使用这些系统调用（也称为进管指令）请求操作系统的服务。一般软件在用户级（又称目态）运行。操作系统在核心级（又称管态）运行。系统调用构成了虚似机的高级指令系统，相当于扩展了裸机的基本指令系统（屏蔽了裸机的特权指令）。在数量上的扩展，是在单机系统中引入了多道程序技术，使多个用户可以同时使用一台计算机，在逻辑上，每个用户都有一台独用的计算机。

2. 资源管理观点

这里的资源是指计算机系统为了进行数值计算和数据处理所需要的物质基础，通常分为系统硬件资源和软件资源。硬件资源是组成计算机和计算机操作所需的物理实体，它们是看得见、摸得着的资源。如：处理器、存储器、输入输出设备（键盘、显示器、打印机、磁盘等）。软件资源是依赖于一定的物理实体才能被人们所感知的一类资源。如：程序、数据等，他们可以经显示器或打印机等设备展现给用户。操作系统是控制和管理计算机系统资源的程序，它的工作是当用户程序和其他程序争用这些资源时，提供有序和可控的分配。

以上分别对操作系统在计算机系统中的位置采用了由顶向下（top-down）和自底向上（bottom-up）考察的观点；另外，还可以从一个程序在系统中的运行过程的角度进行考察。

3. 进程观点

操作系统环境中，常同时运行着多个程序（称为并发），这些同时运行的程序称为“进程”（process）或“任务”（task）。每个进程都完成各自的特定任务（如用户程序的运行、处理某个设备的输入输出等）。操作系统则控制和协调这些进程的运行。操作系统以系统各部分可以并行工作为出发点，考虑管理任务的分割和相互之间的关系，通过进程之间的通信解决共享资源时带来的竞争问题。通常，进程可以分为用户进程和系统进程两大类。

4. 分层观点

操作系统的实现常常采用分层技术，即若干层次的程序按一定的结构组织并协调工作。根据使用环境和对用户作业的处理方式，操作系统的基本类型可以分为三大类：批处理操作系统（batch processing operating system）、分时操作系统（time sharing system）和实时操作系统（real time system）。

分时操作系统使用多个用户同时（称为多路性）以会话方式（称为交互性）控制自己程序的运行，每个用户都感到似乎有一个独立的、支持自己请求服务的系统（称为独立性）。

实时系统往往是专用的，系统与应用很难分离，常常紧密结合在一起。实时系统产不强调资源利用率，更关心及时性（又称时间紧迫性）、可靠性和完整性。实时系统又可分成实时过程控制与实时信息处理两大类。

随着网络技术的发展与普遍使用，共享网络资源的网络环境下的操作系统也已流行。网络环境下的操作系统又分成网络操作系统和分布式操作系统。网络操作系统是在各单机操作系统的基础上，按照网络体系结构的各个协议、标准进行开发，形成网络软件，包括网络管理、通信、资源共享、系统安全和多种网络应用等。分布式操作系统要求有一个统一的操作系统，即废弃（或改造）各单机的操作系统，整个网络设有单一的操作系统。分布式操作系统负责全系统的资源分配和调度，为用户提供统一的界面。用户据此使用系统资源，完成所需任务，至于设备在何处用户是不必知道的，这称为位置透明性（location transparency），是系统透明性的一种，其他还有迁移透明性（migration transparency，资源可不改变名字地移动）、复制透明性（replication transparency，用户不知存在多少个副本）、并发透明性（concurrency transparency，多个用户自动共享资源）、平行透明性（parallelism transparency，用户不知道平行活动的发生）等。分布式操作系统是一个逻辑上紧密耦合的系统，相对应的，网络操作系统用户则需指明欲使用哪一台计算机上的哪个资源。

3.1.2 操作系统的功能

操作系统的职能是管理和控制计算机系统中的全部、软件资源，合理组织计算机工作流程，并为用户提供一个良好的工作环境和友好的接口，所以，系统资源管理和提供用户界面是操作系统的功能要点。计算机的主要硬件资源包括处理器（处理机）、存储器、输入输出设备，软件资源常以文件形式存放在外存储器上，现代操作系统的主要任务也围绕着这五个部分，即相应的五个功能模块。

1. 处理器（处理机）管理

处理器是计算机系统的心脏。在单用户系统或单道系统中，处理器为一个用户或一个作业服务，其管理简单。为提高系统资源利用率，引入多道程序技术，即多个程序（作业）同时运行，它们争用处理器，就要解决对处理器的分配调度策略，实施分配和回收资源。许多操作系统以作业和进程方式进行管理，实现作业和进程的调度，分配处理器，控制作业和进程的执行。现代的操作系统引入了线程（thread）作为分配处理器的基本单位。

基本操作系统对处理器管理策略不同，其提供的作业处理方式也就不同，如简单的批处理方式、分时处理方式、多道成批处理方式和网络环境下的处理方式等。从而，呈现在用户面前的就是不同的操作系统。

2. 存储管理

计算机系统中，存储器（一般称为主存或内存）是运行程序、存放工作数据的设备，存储管理的工作主要是对主存储器进行分配、扩充和保护。系统中有多个程序（操作系统、实用程序和用户程序）共享存储器，它们彼此之间不能相互干扰和破坏，这就是存储保护问题。当计算机系统中运行的程序所需要的主存容量超过比实存（实存内存）大得多的虚存（虚拟存储器），让用户可运行一个比实存大的用户作业，这就是内存扩充问题。

3. 设备管理

现代计算机系统常常配置很多种类的输入输出设备，它们输入输出速度差别很大，计算机系统常常采用通道、控制器、设备三级控制方法管理这些设备，设备管理的任务就是监视这此资源的使用情况，根据一定的分配策略，把通道、控制器和设备分配给请求输入输出操作的程序，并启动设备完成所需的操作。为了发挥设备和处理的并行工作能力，常常采用缓冲技术和虚拟技术。

4. 文件管理（信息管理）

用户使用计算机系统处理数据（信息），这些数据和程序作为文件储存在外部存储器（如磁盘、磁带、光盘等）上，文件管理的任务是管理文件的存储空间，提供信息的共享和保护，允许多个用户协同工作又不引起混乱。

5. 用户接口（工作管理）

前述四项功能是操作对硬、软件资源的管理，操作系统也必须为用户提供一个友好的用户接口——命令接口和图形接口。一般，用户通过两种命令接口请求操作系统的服务。一种接口是作业一级的接口,即提供一组操作命令,如 UNIX 和 Linux 的 Shell 命令语言或作业控制语言（JCL）让用户组织和控制自己作业的运行。作业控制又分成两类：联机控制和脱机控制。另一种接口是编程接口，即提供一组系统调用命令（又称进管指令 SVC）供各种程序（实用程序、应用程序、线程管理）存储服务、设备服务、文件服务、用户界面服务、异常处理服务、其他类型服务等。

3.1.3 操作系统的工作原理

操作系统的基本特征是并发和共享。并发的意思是在许多同时的活动（或并行的活动）：输入输出操作和处理运行并行活动，在主存中同时驻留几道用户程序等都是并发的例子。并发活动会要求共享资源和信息,这就是能提高资源的利用率。多道程序可以并发而共享资源,一个用户的任务也可以组织成几个子任务并发工作而提高运行效率。 一个计算任务的演算模型如型 3.2（a），进一步可描述成图 3.2（b）

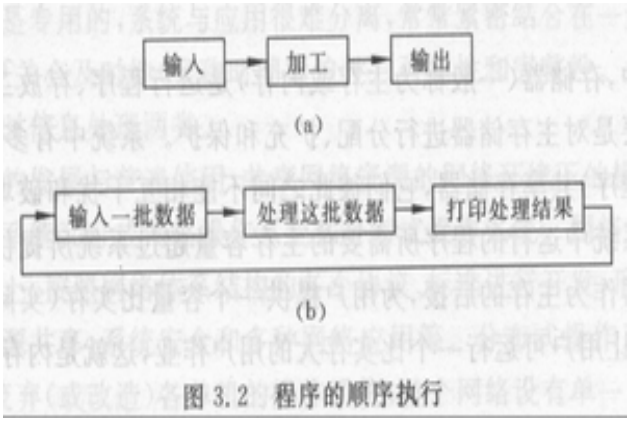


图 3.2 程序的顺序执行

程序这样运行时，不能使输入设备、处理器和打印机并行工作；若忽略处理器加工数据的处理时间，则完成一批数据加工的时间为输入输出设备耗时的总和。若将这个计算任务分成三个子任务，并引入缓冲技术：输入子任务从输入设备读一批数据到输入缓冲缓冲区，处理子任务则把输入缓冲区中的数据处理后放入输出缓冲区，打印子任务则打印输出间近似于系统中较慢的设备的速度。这一并发执行的过程如图 3.3 所示。

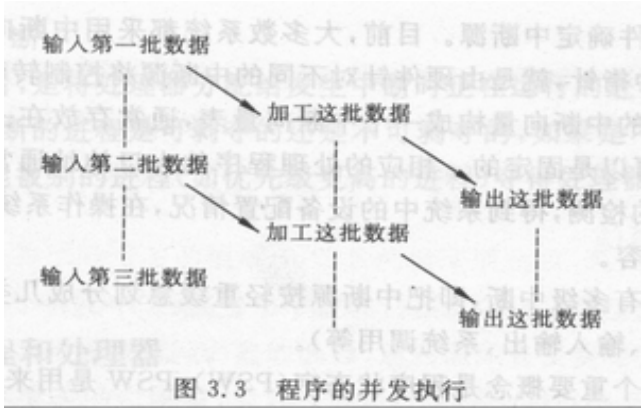


图 3.3 程序的并发执行

程序的并发执行，发挥了处理器与输入输出设备并发工作的能力，提高了系统的效率。多个计算过程同时驻留在主存储器中并发执行，这种程序设计方法称为“多道程序设计”这种系统称为多道程序系统。多道程序系统能充分发挥处理器的利用效率，提高系统资源的利用效率。既然它们同处在一个系统中，一个任务（或子任务）的执行就会受到其他任务（或子任务）的影响（又称为制约）。

3.1.4 操作系统的硬件基础

现代计算机系统设备的发展异常迅速，除了拥有常用的键盘、显示器、磁盘驱动器、鼠标、光盘驱动器等，还可配有数字化仪、声频设备、网络设备、办公设备等。这些设备的加入，导致了设备控制技术的发展，设备控制的基本技术是中断。

1. 中断的基本概念

中断的基本概念是打断正在进行的工作，转向处理更紧急的事件，处理完成后再继续原来的工作。在计算机系统中，中断是改变处理器执行指令顺序的事件，被打断的程序可以在将来某个时候（可能在中断处理完成后立即或推迟，这是由操作系统控制的）恢复执行。

计算机系统的中断既来自系统内部，也来自系统外部，分别称为内部中断和外部中断。

（1）内部中断是系统本身在工作过程中出现的各种需要紧急处理的事件：一是由于运行程序发生意外而产生的，如溢出、操作地址错误等；另一种是运行程序需要发生的，如系统调用。内部中断也称为软件中断。

（2）外部中断是由于硬件方面的原因，故也称为硬件中断，如掉电、设备运行完成、设备故障、时钟中断等。硬件中断往往是随时发生的，不是由正在运行的程序所控制的。外部中断和由于运行程序产生的例外中断常常称为强迫中断，而运行程序主动要求而产生的中断称为自愿中断。外部中断可进一步分成可屏蔽中断和不可屏蔽中断。不可屏蔽的中断是一些最紧急、最重要的中断，如掉电等；可屏蔽中断通过处理器内部的中断许可状态确定响应中断的次序，或者不响应一些不重要（或不紧急）的中断请求。

2. 中断的响应与处理

引起中断的事件（即原因）称为中断源。确定中断源的难易程度与硬件有关。早期的计算机硬件非常简单，各种中断都将控制转移到同一个处理单元（称为中断处理程序入口地址），进一步再由软件确定中断源。目前，大多数系统都采用中断向量的技术。所谓中断向量，实际上是一种指针，就是由硬件针对不同的中断源将控制转移到不同的中断处理程序入口地址。所有的中断向量构成一个中断向量表，通常存放在一个专门的存储区域中，这个区域的地址可以是固定的。相应的处理程序的入口地址通常是在系统引导过程中，通过对系统配置的检测，得到系统中的设备配置情况，在操作系统装入过程中，填写中断向表中的指针内容。

现代计算机都设有多级中断，即把中断源按轻重缓急划分成几类（如硬件故障、程序违例、重新启动、外部、输入输出、系统调用等）

中断响应的另一个重要概念是程序状态字（PSW），PSW 是用来控制指令执行顺序并且保留和指示与程序有关的系统状态。一般，PSW 包含三个部分内容：程序基本状态（指令地址、条件码、处理器状态等）、中断码（程序运行时，当前发生的中断事件）、中断屏蔽位（指出程序运行中发生中断事件时，是否响应）。每个程序都有一个程序状态字描述程序的运行状态。一个处理器有一个用来存放当前运行程序的 PSW 的寄存器（程序状态字寄存器）。为了说明中断响应过程，先分区分三种 PSW：

（1）现行 PSW：当前正占用处理器的程序的 PSW；

（2）新 PSW：出现中断事件后，由操作系统的中断处理程序占用处理器，这个中断处理程序的 PSW 称为新 PSW；

（3）老 PSW：被中断处理程序中断运行的程序的 PSW。

这样，中断响应的过程可描述成图 3.4。图 3.4 中①表示中断机制发现中断事件并允许响应这类中断时，在现行 PSW 中记录中断原因；接着执行②，保存现行 PSW 到这类中断的老 PSW 中；然后执行③，把这类中断的新 PSW 装入到现行 PSW 中。这三个动作是由中断机制自动实现的。当交换 PSW 的硬件操作完成后，现行 PSW 装中包括有关中断类型的中断处理程序的入口地址，然后由中断处理程序处理这个中断。

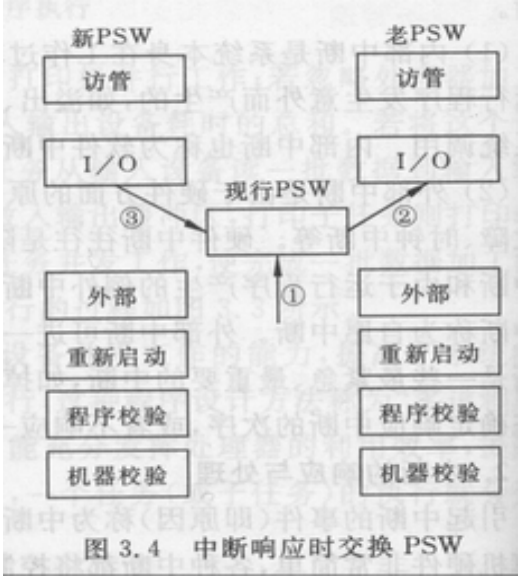


图 3.4 中断响应时交换 PSW

中断处理程序对中断事件的处理分两步进行：

（1）保护好被中断程序的现场信息，即保存被中断程序的寄存器以及 PSW 的内容，以保证中断程序以后能继续运行；

（2）具体处理中断

中断处理完成后，是将处理器分配给发生中断时正在运行的进程。还是分配给另一个进程，这取决于被中断的进程是可剥夺的还是不可剥夺的，如果是不可剥夺的，它重新得到处理器；否则，可能被别的进程（如优先级更高的进程）夺得处理器。

3.2 并发进程

3.2.1 程序、进程和处理器

在计算机系统上运行的是程序。程序是指令的集合，它完成确定的任务。在只允许一个程序运行的系统（称为单道系统）中，这个程序独占系统资源，而系统按程序的指令顺序进行。程序的顺序执行有两个基本特征，即程序的封闭性和程序的可再现性。封闭性是指程序运行时独占系统资源，只有程序本身能改变系统的状态；可再现性是指程序运行不受外部影响，只要初始条件相同，运行结果相同。这两个特征使系统的管理简单，程序员也能够调试程序，然而，系统资源利用率会很低。

多道程序系统让多个程序在系统中轮流运行，当一个程序不用处理器时，另一个程序就使用。也就是就，处理器在程序间切换，从而获得宏观上的并行（微观上的串行），以提高处理器的利用率。这种切换通常是由中断引起的。由于中断是以不可预测的次序发生，即程序的指令执行序列也不可预测的次序前进，这样就会产生操作系统的另一特性：不确定性。即在多道程序系统中，顺序程序的封闭性和可再现性消失了。

【例 3.1】用两个程序完成交通流量的统计以及每一小时报告一次准点时运行的工作，程序如下：

```
统计程序                          报告程序
    loop                              loop
       if 有车通过 then          S2：PRINT COUNT;
S1：COUNT：=COUNT+1;              S3：COUNT：=0;
                                        WAIT（ ）;
     Endloop                           endloop
```

可以看出，当这两个程序在准点时刻并发运行时，标号 S1，S2 和 S3 这三条语句有三种执行顺序：

S1，S2，S3；

S2，S3，S1；

S2，S1，S3。

显然，三种执行顺序的结果不同的；更有甚者，第三种情况将使统计数值出现误差。这显示出在多道程序环境中，程序与程序的执行之间的对应关系发生了变化，需要采用一个新的概念——进行赤描述程序的执行：进程运行中的程序。

3.2.2 进程间的通信

在操作系统中，进程是可以独立运行的程序单位，系统中的各个程序并不是相互隔离的：一方面它们相互竞争使用有限的资源，如处理器、存储器、设备或文件等；另一方面，他们可能相互协作以达到用户作业预期的目的。因此，进程之间需要某种形式的通信。进程之间直接发生联系的称为相交进程（又称相关进程），

否则称为不相交进程。相交进程需要协调彼此间的相对运行速度，存在着直接制约的关系，不相交进程之间的联系将表现在资源使用上的间接制约关系。

1. 互斥

在多道系统中，各进程可以共享各在资源，但有些资源却一次只能供一个进程使用。这种资源常常称为临界资源，如打印机，公共变量、表格等。

互斥是要保证临界资源在某一时刻只被一个进程访问。

2. 同步

一般，一个进程对于另一个进程的速度是不可预测的，也就是说，进程之间是异步运行的。这了成功地协同工作，有关进程在某些确定的点的上应当同步他们的活动：一个进程到达了这些点后，除非另一进程已经完成了某个活动，否则就停下来，以等待该活动结束。

3. 死锁

当若干个进程竞争使用资源时，可能产生下述情况：每个进程要求资源都已被另一进程占用，于是也就没有一个进程能继续运行。这种情况称为死锁。

3.2.3 信号量与 P，V 操作

用顺序程序设计处理同步与互斥是十分困难的，为此引入信号概念：信号量是一种特殊的变量，这的表现形式是一个整型变量及相应的队列；除了设置初值外，对信号量只能施加特殊的操作：P 操作和 V 操作，P 操作和 V 操作都是不要分割的原子操作，因此也称为原语，它们定义分别为：

P（S）::=S：=S-1

if S＜0then

调用者（进程改变状态，转相应队列等待。）

V（S）::=S：=S+1

if S≤0then

从相应队列中选进程，唤醒它。

P 操作的作用是将信号量 S 的值减 1，若 S 的值变成负数，则调用 P 操作的进程暂停执行。直到另一个进程对同一信号量作 V 操作。V 操作的作用是将信号量 S 值加入 1，若 S 的值小于等于 0，则从相应队列（关于 S 的队列）中选一个进程并唤醒它。

P，V 操作也可记为 down（）和 up（）或 wait（）和 signal（）。

如果 P，V 操作不是原子操作，就可能出现与时间有关的错误。例如，多个进程对同一个初值为 1 的信号量作 P 操作，就可能连续减几次，从而使原来可以有一个进程继续运行的状态，变成都必须等待的状态。

为了保证，P，V 操作是原子操作，需要硬件的支持：可以采用屏蔽中断的方法，或利用特殊的硬指令，如指令 TdstAndset（i）和 Exchange（r，m）等。

1. 互斥

为了保护共享资源（如公共变量等），使它们不被多个进程同时访问，就要阻止这种进程同时执行访问这些资源的代码段，该代码段称为临界区，这种资源称为临界资源。对临界资源访问的互斥要可视作执行临界区的互斥要求。 互斥临界区的管理要求是：

- 有空即进；
- 无空则等；
- 两者择一；
- 有限等待。

利用信号可以方便地实现互斥临界区的管理要求。例如，令信号量 mutex 的初值为 1，于是临界区就改写成下列形式的代码段：

```
P（mutex）；
临界区
V（mutex）；
```

由于 mutex 初值为 1，P、V 是原子操作，因此确实可以实现互斥。

可以看出，进程要访问共享变量或共同资源（临界资源）时，可利用信号量保证正确。

【例 3.2】交通流量统计任务可以改写成：

```
统计程序                          报告程序
loop                              loop
if 有车通过 then
begin                             P (mutex);
p (mutex);                        PRINT COUNT;
COUNT: =COUNT+1;                  COUNT: =0;
V (mutex);                        V (mutex);
End                               wait (60*60);
Endloop                           endloop
```

这里信号量 mutex 的初值为 1。

2. 同步

最简单的同步形式：进程 A 在另一进程 B 已到达 L2 以前，不应前进到超过点 L1。对此，可以令信号量 proceed 初值为 0，实现这种同步的程序形式是：

```
进程 A                            进程 B
…                                 …
L1: P (proceed);                  L2: V (proceed);
…                                 …
```

更复杂一点的同步问题是单缓冲区的生产者-消费者问题。

【例 3.3】单缓冲的生产者-消费者问题。

要求存后再取，取后再存，即有两个制约关系，为此，需要两个信号量，记为 empty 和 full，它们的初值分别是 1 和 0，相应的程序段形式是：

```
生产者
loop
生产一个产品 next;
p (empty);
next 存缓区;
V (full);
…
endloop
```

```
消费者
loop
p (full);
从缓冲区中取一个产品;
V (empty);
享用;
…
endloop
```

上述程序段也可改写成：

```
生产者
loop
…
生产一个产品 next:
P (empty);
P (mutex);
Next 放入缓冲池;
V (mutex);
V (full);
…
endloop
```

```
消费者
loop
…
p (full);
p (mutex);
从缓冲池中取出一个产品
V (mutex);
V (empty);
享用;
…
endloop
```

注意，此时，信号量 empty 和 full 的初值都是 0。

信号量 S 的值是有物理意义的：当 S>0 时，表示有 S 个可用资源，即有 S 个进程能依次通过相应的 P 操做。当 S<0 时，表示有|S| 个进程正在相应的队列上等待。

3. 同步与互斥

对于多个缓冲区（缓冲池）的管理，由于存入缓冲区和从缓冲区取出是对缓冲池的操做，因此，需要互斥。

【例 3.4】假设缓冲池中有缓冲区 100 个，相应的生产者-消费者问题的程序是：

```
生产者
loop
    …
    生产一个产品next;
    p (empty) ;
    P (mutex) ;
    next放入缓冲池;
    V (mutex) ;
    V (full) ;
      …
    endloop
```

```
消费者
loop
     …
     p (full) ;
      P (mutex) ;
      从缓冲池中取出一个产品
    V (mutex) ;
      V (empty) ;
      享用;

      endloop
```

这里，信号量 empty，full 和 mutex 的初值分别是 100，0 和 1。

注意，上述程序中的两个 P 操作是不可互换的，若互换会引起与时间有关的错误，有时会引起死锁。

可以看出，用信号量解决同步与互斥问题是复杂的，如果 P，V 操作不成对，相邻的两个 P 操作互换次序将会产生问题。

3.2.4 高级通信原语

P，V 操作是用来协调进程进的关系的，编程较困难而且没有信息交换，故常称为低级通信原语。此外，P、V 操作必须有共享存储器，而高级通信原语则提供两种通信方式：有缓冲区的通信和无缓冲区的通信。

（1）有缓冲区通信原语：

write（Buffer-Name，Varible）等缓冲区空再存入

Read（Buffer-Name，Variable）等缓冲区满再取出

（2）无缓冲区的通信原语，又称为消息传递：

Send（Who，Message）发送消息给指定进程一组进程

Receive（Who，Message）从约定进程接收消息

有的系统还提供带标记的发送，有

Send（Who，Message，Tag）

用 Tag 可指定发送进程是否要等待接收进程取到内容以后再继续运行。一般，接收进程总果等待消息到达后才继续运行。

3.2.5 管程

管程是一种并发性的构造，它包括用于分配一个特定的共享资源或一组共享资源的数据和过程。为了完成分配资源的功能，进程必须调用特定的管程入口。许多进程可能打算在不同的时间进入管程，但在管程边界严格地实施互斥，在某一时刻，只允许一人进程的人，当管程中已有一个进程时，其他希望进入管程的进程必须等待。这种等待是由管程自动管理的。因为保证了互斥，所以可降低因使用 P，V 操作不当而发生问题（例如产生不确定的结果）的概率。

管程中的数据或者是管程中所有过程的全局变量，或者是某个特定过程的局部变量。所有这些数据只能在管程内访问，在管程外的进程无法访问管程内的数据。这叫做信息掩蔽——一种系统结构技术，它极大地便利了研制更可靠的软件系统。 如果调用管程入口的进程发现资源已被分配，管程呼这个入口对应的过程

调用等待（Cwait）操作。这个进程可以仍留在管程中，但若另一进程又进入管程，这就会违反互斥原则。因此调用 Cwait 操作的进程外等待资源被释放。

最终，占用资源的进程将调用一个管程入口以把资源归还给系统，这个入口可能仅只是接受被归还的资源，等待另一个申请分配资源的进程到来。这时，如果有等待分配这个资源的进程，这个管程入口过程将调用发信号（Csignal）操作以允许其中一个等待的进程重新进入管程，获得资源并离开管程。如果一个进程入口过程调用 Csignal 操作，这时没有进程在等待分配资源，那么这人发信号操作不产生什么结果（当然，管程得到资源）。很明显，等待分配资源的进程必须在管程外等待，以允许另一个进程进入管程释放资源。管程结构如图 3.5 所示。

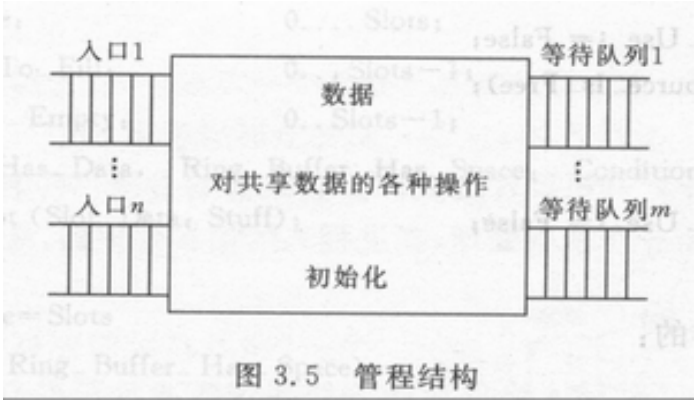


图 3.5 管程结构

为了保证等待分配资源的进程最终一定能够得到它，管程优先为已经等待分配资源的进程分配资源。而不是为新到达的想进入管程获得资源的进程分配资源。否则，新到达的进程可能在等待进程再一次进入管程前“抢”到资源。如查这种不幸的序列真的反复出现的话，等待再次进入管程的进程将会被无限延迟。

实际上，由于各种原因，进程希望（必须）在管程外等待，因此引入了条件变量这个概念。每个独立的条件变量是和进程可能需要等待的不同原因相联系的。等待操作和发信号操作需要修改，以包括它们各对应的条件变量：

Cwait （Conditional_Variable_Name）

Csignal（Conditional_Variable_Name）

条件变量和我们熟悉的“普通的”变量是非常不同的。当定义一个条件变量时，就建立了一个队列。调用等待操作（Cwait）的进程被连到这个队列上，调用发信号（Csignal）操作的进程将一个等待进程从队列中摘下并进入管程。

【例 3.5】管程实例 1。

假设几个进程都需要访问在某个时刻只能供一个进程使用的资源。一个管理分配和归还上述资源的简单管程是

Motitor Resource_Allicator;

```
Var  Resource_In_Use： Boolean;
         Resour_Is_Free： Coneition;
Procedure Get_Resource;
Begin
             If    redource_In_Use Then
                       Cwait（Resource_Is_Free）;
             Resource_In_Use：=True
End;
Procedure Return_ Resource;
Begin
              Resource_In_Use：=False;
              Csignal（Resource_Is_Free）;
End;
Begin
               Resource_ In_ Use：=False;
End;
```

用户的程序是这样的：

```
┇
Resouce_Allocator.Get_Resource;
```

用资源：

```
Resouce_Allocator.Return_Resource;
```

管程 Resource_Allocator 的妙处在于它精确地像二元信号量一样工作，Get_Resource 的功能和 P 操作一样，Return_Resource 的功能和 V 操作一样。这样简单的一个资源的管程可以用于实现信号量，因此管程至少和信号量一样有用。

【例 3.6】管理实验 2。

当代的操作系统一般是由一组在内核控制下的异步并发进程实现的。这些进程独立地管理并行活动，但偶尔也需要互相通信。这里讨论环形缓冲器，说明在生产者进程向消费者进程传递时如何使用环形缓冲器。

生产者可能在消费者还不需使用数据时产生数据，即超前生产，消费者也可能在生产者还未产生出数据时消费数据。因此，在生产者和消费者间的同步是很重要的。

操作系统通常允许把了些固定数量的存储单元用于生产者和消费者进程间的通信,这可以用一个指定大小的数组来模拟。生产者把数据存入数组连续的元素台，消费者按这些数据存入的顺序把它们取出。生产者可能比消费者领先几个元素。最终生产者把数据放入数组的最后一个元素中。当产生下一个数据时，它必须“绕回”并从数组的第一个元素开始存放数据（当然应该假定消费者已经把生产者先前放在那里的数据取走）。这个数组实际上闭合成一个环，因此称为环形缓冲器。 因为环形缓冲器的大小是固定的，生产者偶尔会发现所有的数组元素都已填满，此时必须使生产者等待，直到消费者取出一个数组元素为止。类似地，也会出现消费者打算取出数据而数据组空着的机会,这种情况下必须使消费者等待，直到生产者把数据放入一个数组元素为止。管程 Ring_Buffer_Monitor 实现了一个环形缓冲器和处理生产者-消费者关系的同步机构。

```
Monitor Ring_Buffer_Monitor;

Var     Ring_Buffer;      Arry[0...slots-1] Of Stuff;

            Slot_In_Use:    0...slots;

            Next_Slot_To_Fill ;  0...slots-1;

       Next_Slot_To_Fill;  0..slots-1;

       Ring_Buffer_Has_Data, Ring_Buffer_Has_Space : Condition;

Procedure Fill_A_Slot (Slot_Data : Stuff)

Begin

       If Slots_In_Use=Slots

             Then Cwait (Ring_Buffer_Has_Space);

       Ring_Buffer[Next_Slot_To_Fill]:=Slot_Data;

       Slot_In_Use :=Slot_In_Use +1;
```

```
        Next_Slot_To_Fill :=(Next_Slot_To_Fill +1)Mod Slots;
        Csignal(Ring_Buffer_Has_Data);
End ;
Procedure Empty_A_Slot(Var Slot_Data:Stuff);
Begin
        If Slot_In_Use=0
              Then Cwait(Ring_Buffer_Has_Data);
         Slot_Data :=Ring_Buffer[Next_Slot_To_Empty ];
         Slot_In_Use :=Slot_In_Use-1;
         Next_Slot_To_Empty :=(Next_Slot_To_Empty +1)Mod Slots;
         Csignal(Ring_Buffer_Has_Space);
End ;
Begin
         Slot_In_Use               :=0
         Next_Slot_To_Fill   :=0;
         Next_Slot_To_Empty :=0;
End ;
```

我们假设这个数组包含 Slots 个数据类型为 stuff 的元素（也叫做存储槽）。变量 Next__Slot__To__Fill 指出产生的下一个数据项放在环形缓冲器的什么位置，变量 Next__Slot__To__Empty 指出下一个数据项从什么位置取出。当生产才发现环形缓冲器完全装满时，在条件变量 Ring__Buffer__Has__Space 的队列上等待；当消费者取出一个存储槽中的数据时，通知在这个队列上等待的生产者。当消费者发现环形缓冲器变空时，在条件变量 Ring__Buffer__Has__Data 的队列上等待；当生产者在存储槽中存放完一个数据时，通知在这个队列上等待的消费者。

3.3 系统的核心

操作系统核心位于计算机硬件之上。核心的目的是提供一种进程赖以生存的环境，其主要任务是接收中断并作基本处理，在进程之间切换处理器，处理进程之间的通信。

3.3.1 基本硬件机构

（1）中断机构：为了使输入输出活动理处理活动并行进行，当一台外部设备的传输操作结束时，应能中断正在运行的进程，由中断处理程序处理。

（2）特权指令：为使各并发进程不相互干扰，系统的部分指令集应保留仅供操作系统使用。这些指令称为特权指令，如允许与禁止中断、执行输入输出操作等。为了区分什么时候可以使用特权指令，什么时候不可以使用，系统至少具有两种工作状态，它们分别称为系统态和用户态，又称为管态和目态。特权指令只能在管态下运行。

（3）主存保护：多道程序系统中，必须对各进程使用的主存加以保护，以防止其他进程进行未经许可的操作。保护机制应包含在主存对址硬件中。

（4）时钟：硬件时钟以固定的时间间隔产生中断信号，这对于实现处理器的调度以及实现与时间有关的任务是不可缺少的。

3.2.2 进程

1.进程

进程是一个程序关于某个数据集的一次运行。也就是说，进程是运行中的程序，是程序的一次运行活动。相对程序，进程是一个动态的概念；而程序是静态的概念，是指令的集合。因此，进程具有动态性和并发性。 在操作系统中进程是系统资源分配、调度和管理最小的单位，注意，现代操作系统中还引入了线程，它是处理器分配的最小单位。

2.进程的状态及其转换

多道系统中，进程的运行是走走停停的，它在处理器上的交替运行，使它的运行状态不断地变化着，最基本的状态有三种：运行、就绪和阻塞。

- 运行（running）：正占用处理器；
- 就绪（ready）：只要获得处理器即可运行；
- 阻塞（blicked）：正等待某个事件（如 I/O 完成）的发生。

现代操作系统中还有挂起（suspend）状态。 进程的状态及转换如图 3.6 所示。

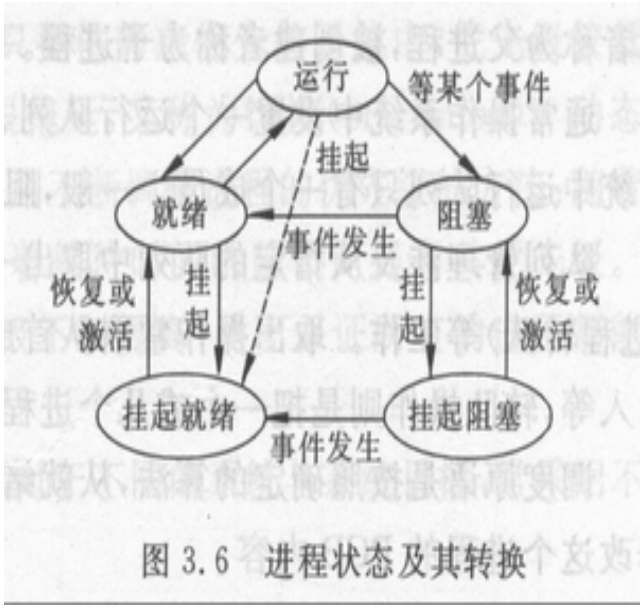


图 3.6 进程状态及其转换

3.进程控制块

进程是一个动态的概念，如何表示一个进程？在操作系统中，引入数据结构——进程控制块（PCB）标记进程。PCB 是进程存在的惟一标记，PCB 描述了进程的基本情况。从静态的观点看，进程由程序、数据和进程控制块组成；从动态的观点看，是计算机状态的一个有序集合。

程序是进程运行所对应的运行代码，一个进程对于程序；有的程序可以同时对应于多个进程，这个程序的代码在运行过程中不会被改变，常称为纯码程序或可重入程序，他们是可共享的程序。

进程控制块保存进程状态、进程性质（如优先程度）、与进程有关的控制信息（如参数、信号量、消息等）、相应对列和现场保护区域等。进程控制块随着进程的建立而产生，随着进程的完成而撤消。图 3.7 是一个简化的 PCB 示意图。

进程标识	状态	优先级	控制信息	队列	访问权限	现场

图 3.7 一个简化的 PCB

PCB 是操作系统核心中最主要的数据结构之一，它既是进程存在的标志和调度的依据，又是进程可以被打断并能恢复运行的基础。核心通过 PCB 管理进程，一般，PCB 是常驻内存的，尤其是调度信息必须常驻内存。有的系统为降低主存使用量（即降低系统开销），把 PCB 分成常驻内存部分和可以交换（入/出内存）两部分。

3.3.3 进程管理

在操作系统中有许多进程，它们对应着不同的或相同的程序，竞争地使用着系统的资源。进程管理涉及到进程控制、队列管理、进程调度等。

1.进程控制

进程的生命过程是从它被创建的开始，直至任务终止而撤消。其间会经历各种状态的转换，它们都是在操作系统控制下完成的，为此，操作系统提供了对进程的基本操作，也称为原语。这此原语包括：创建原语（create）、阻塞原语（block）、挂起原语（suspend）、恢复原语（resume）、终结原语（terminal）、优先级原语（chang_ priority）和调度原语（schedule）。它们可以被系统本身调用，有的也以软中断形式供用户进程调用。它们都涉及现场队列管理等。例如，创建原语是调用命令，供用户进程调用。在命令处理中涉及许多操作：分配一个 PCB，赋予标识数，分配初始优先数，分配初始资源，把这个 PCB 插入相关队列（如绪队列，有的系统放入挂起就绪队列）。当一个进程创建一个新的进程时，创建者为父进程，被创建者称为子进程。

通常操作系统中设置一个运行队列、一个就绪队列和若干个阻塞队列。在单处理器系统中运行队列只有一个成员。一般，阻塞队列的个数取决于等待事件（原因）的个数。

队列管理涉及从指定的队列中取出一个成员（进程），把这个进程插入另一指定队列（进程转队）等工作。取出操作常取队首成员，插入操作可以是插入队首、队尾或按优先数插入等，转队操作则是把一个或几个进程从一个队列中取出而插入另一个队列。 调度原语是按照确定的算法，从就绪队列中选择一个就绪进程，将处理器分配给它，修改这个进程的 PCB 内容。

在操作系统中进程生命的简单活动是这样的：一个进程可以由系统创建（如用户运行一程序），也可以由用户进程用创建原语建立。新建立的进程开始时处于就绪状态或挂起就绪状态。对后者，操作系统将在条件许可时，采用恢复或激活原语使其成为就绪状态，即处在就绪队列中，直到被调度原语选中而占有处理器。该进程的运行也会因等待某个事件（如 I/O 完成）的发生而出于阻塞，转入相应的阻塞队列。一旦相应事件发生后，将被唤醒原语叫醒而回到就绪队列。重复上述过程直至运行完毕，经终止原语作一些簿记工作并撤消这个进程。

2.进程调度

进程调度即处理器调度，它的主要功能是确定在什么时候分派处理器，并确定分给哪一个进程。在分时系统中，一般有一个确定的时间单位（称为时间片），当一进程用完一个时间单位时，就发生进程调度（又称上下文转换），即让正在运行的进程改变状态并转入就绪队列队尾，再由调度原语将就绪队列的首进程取出，投入运行。

进程高度的方法基本上分为两类：剥夺调度与非剥夺调度。所谓非剥夺调度是指一旦某个作业或进程占有了处理器，这个进程就占用处理器直到主动放弃处理机为止；相反，如果其他进程可获得处理器的使用权则是剥夺调度。

有些进程（如高优先级进程）需要快速服务的系统中，剥夺调度是非常有用的。例如，在实时系统中丢失一个中断的后果将是灾难性的。交互式系统中，为保证用户可接受的响应时间，剥夺调度也十分重要。

因为剥夺要付出代价，上下文转换也有开销，所以要使剥夺调度有效，主存中必须保持多个进程。以便当一个进程可以获得处理器时，它已处于就绪状态并在主存。要将当前不运行的进程保持在主存，也是有开销的。

在非剥夺系统中，短的作业常常等待长的作业，但所有进程受到的服务是公平的，响应时间是可以预测的。

对就绪队列处理体现了调度策略，常用的是按优先级处理。有两种确定优先级的方法，即静态和动态。静态是指进程的优先级在进程开始运行前确定，运行过程中不变；动态优先级则可以在进程运行过程中改变。在只提供静态优先的系统中，优先级较低的进程可能迟迟得不到处理器的服务，甚至被无限拖延。例如，UNIX系统采用不断调整进程的优先级的方法：随着占用处理器时间的延长而降低其优先级，反之，随着等待时间的延长而提高其优先级。这样，随着等待时间的延长，就绪状态进程的优先级不断提高，从而保证所有的进程都能获得处理器的服务。

进程调度的算法是服务于系统目标的策略，对于不同的系统与系统目标，常采用不同的调度算法，如：

- 先来先服务（FCFS）：又称先进先出（FIFO），就绪队列按先来后到原则排队。

- 优先数调度：优先数反映了进程优先级，就绪队列按优数排队。

- 轮转法（round robin）：就绪队列按 FCFS 方式排队。每个运行进程一次占有处理器时间都不超过规定的时间单位（时间片），若超过，即调用调度原语而形成就绪进程轮流使用服务器的事实。

3.4 存储管理

现代计算机系统中的存储系统常是多级存储体系，至少有主存（内存）和辅存（外存）两级，有的系统有更多级。主存是由系统实际提供的存储单元（常指字节）组成的一个连续地址空间，处理器可直接存取。辅存是指如软盘、硬盘、光盘和磁带等一些外部存储部件，常用来存放暂不执行程序和数据。处理器不能直接访问外存，需通过启动 I/O 设备，才能进行内、外存交换。外存访问速度慢，但价格便宜，常用作主存的后援设备。

主存大小由系统硬件决定，是实实在在的存储，它的存储容量受到实际存储单元的限制。

虚拟存储（简称虚存）不考虑实际主存的大小和数据存取的实际地址，只考虑相互有关的数据之间的相对位置，其容量由计算机的地址的位数决定。

系统中主存的使用一般分成两部分：一部分为系统空间，存放操作系统本身及相关的系统数据；另一部分为用户空间，存放用户的程序和数据。

1. 地址重定位

用户程序需调入主存运行，即从辅存把用户已经编译链接的目标程序（有时称为可执行程序）装入主存。由于用户作业的存储空间是运行时确定的，所以程序中的操作地址都采用相对地址（逻辑地址）的形式。把相应地址空间的程序转换成在绝对地址（物理地址）空间上能够执行的过程称为地址重定位，也称为地址映射或地址映像。地址重定位有两种：静态重定位和动态重定位。静态重定位是指在程序装入时完成。一般由软件实现；动态重定位是指在程序执行时实现地址转换，它通常由一个基本地址寄存器和一个加法器组成的动态重定位机制实现，图3.8是其示意图。

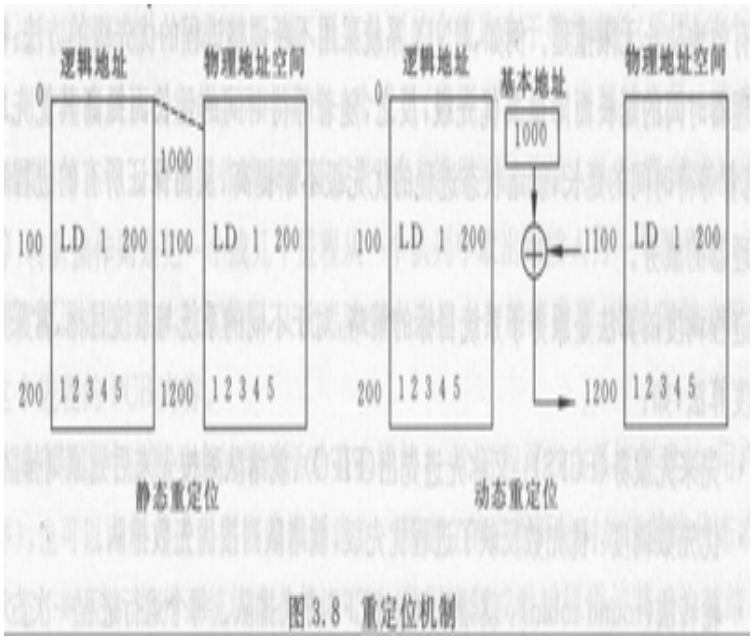


图3.8 重定位机制

2. 存储管理的功能

早期的单用户操作系统，一次只允许一个用户程序驻留，它拥有用户地址空间的全部访问权限，存储管理的任务是存储空间的分配与回收。

在多道程序系统中，多个程序同时驻留内存；需要解决如何有效地利用主存，如何让需要较大运行空间的作业运行，如何保护与共享主存等问题。存储管理要提高存储资源的利用效率，同时又方便用户使用，因此应具有如下功能：

（1）分配与回收：主存分配方法有两种：静态和动态。静态分配是指在目标模块装入主存时即取得所需空间，直至完成不再变动；动态分配则允许进程在运行过程中继续申请主存空间。

采用动态分配方法的系统中，常配合使用合并自由区的方法，使一个空区尽可能地大。

（2）存储扩充：提供虚拟存储器，使计算机系统似乎有个比实际主存储器容量大的主存空间。需考虑放置策略、调入策略和淘汰策略。

（3）共享与保护：共享指共享在主存中的程序或数据，如多个用户共享编辑程序或编译程序等。由于多道程序共享主存，每个程序都应有它单独的主存区域，各自运行，互不干扰。 常用的主存保护方法有硬件方法、软件方法和软硬件结合方法。硬件方法有界地址保护法和存储访问键方法。界地址保护法是为每个进程设置上下界地址，其内容就是程序段和数据的起始地址和终止地址，用于对访问地址作合法性检查（如越界否）。存储访问键是为每一个受保护的存储块分配一个保护键，而不同的进程有不同的权限代码，仅当权限代码与存储保护匹配时才允许访问，否则将产生访问违例中断。

3.4. 1 实存管理

1. 单一连续分配

在单道程序系统中，主存区域用户空间全部为一个作业或进程占用，如图 3.9 所示。单一连续分配方法以主用于早期单道批处理系统以及 80 年代个人计算机系统。单一连续分配方法主要采用静态分配方法，为降低成本和减少复杂度，常不对主存进行保护，因此会引起冲突而使系统瘫痪。

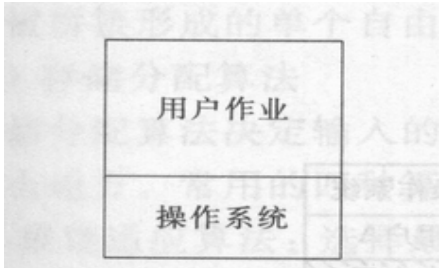


图 3.9 单一连续分配

2. 固定分区分配

固定分区分配是把主存空间固定地划分为若干个大小不等的区域，划分的原则由系统决定。系统使用分区表描述分区情况，如图 3.10 所示。

分区起始地址	分区长度	状态
100	50	已用
150	100	自由
200	150	已用
400	200	自由

图 3.10 固定分区分配

3. 可变成分区分配

可变分区分配方法是将主存空间按用户要求动态地划分成若干个分区，这样就消除了固定分区分配方法中的小作业占据大分区造成的浪费（这种浪费称为内碎片）。可变成分区分配系统中初始时只有一个分区。随后，分配程序将这个区集资分给作业或进程，如图 3.11 所示。



图 3.11 可变分区分配

再考察连续分配方案：一个作业必须占据相邻接的存储单元/在可变分区分配系统中，并不作出作业有多长的假定（除了他们不能大于计算机内可利用主存的尺寸之外）。当作业到达时，如果调度机构决定他们开始运行，他们就能获得必要的存储区，一点浪费也没有——存储区的分区与作业的长度相符。

但是，每种存储组织方案都包含一定程度的浪费。在可变分区分配系统中，主存中的作业在开始装入和归还自由区之前。主存浪费并不明显，如图 3.12 所示。这些自由区可以被其他作业使用。即使如此，剩余的自由区域可能变得很小。因此在可变分区分配系统中，确实会出现存储器浪费的现象（称为外部碎片）。

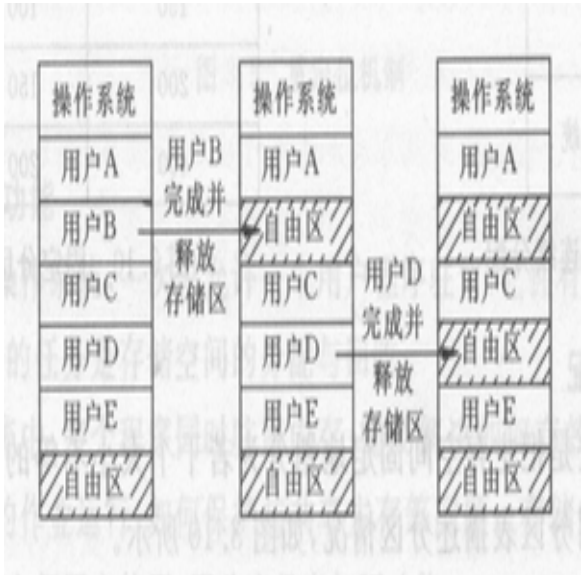


图 3.12 可变分区分配系统中的“自由区”

（1）合并自由区

在可变分区分配系统中，当一个作业完成时，能够检测到被释放的存储区是否与其他自由存储区域（自由区）相邻接。如果与其他自由存储区邻接，可以在自由存储区表记录上新增加一个自由区，或将新的自由区合并为单一自由区。

合并相邻接的自由区以形成单个更大的自由区的过程叫作合并，由图 3.13 说明。用合并自由区的方法，能够重新获得最大可能连续的存储块。

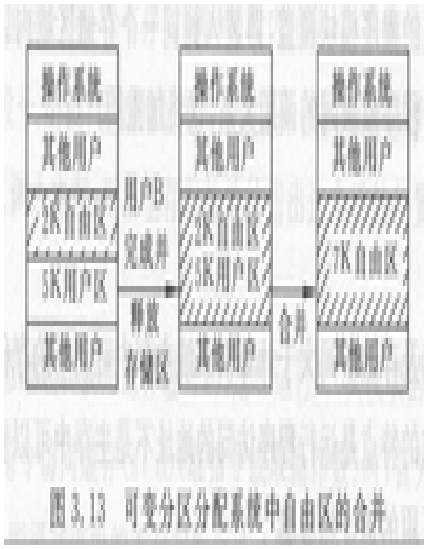


图 3.13 可变分区分配系统中自由区的合并

（2）存储拼接

即使合并了自由区，还是经常发现分布在主存各处的破碎的自由区在主存中占据了相当数量的空间。有时，一个作业申请一定数量的主存，而此时却没有单个的自由区大到足够装下这个作业，显然自由区的总和大于新作业所需要的存储区。

存储拼接或存储紧凑（如图 3.14）也称碎片收集，即移动存储器的所有被占用区域到主存的某一端。这样可以形成大的存储自由区，取代在可变分区多道程序设计中常见的许多小自由区。当所有可利用的自由存储区连续时，一个正等待着的作业能够调入运行，因为其存储需求能被拼接形成的单个自由区所满足。

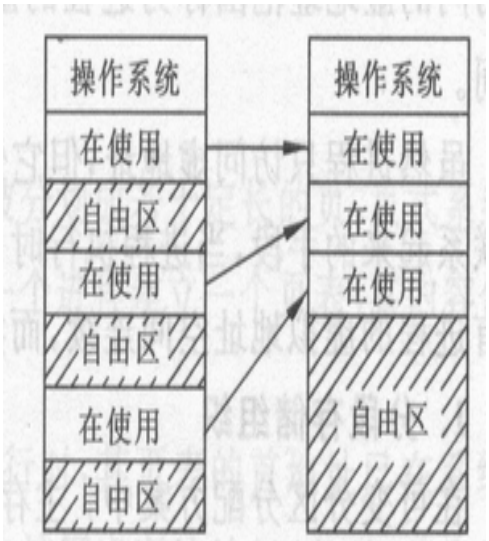


图 3.14 可变分区分配系统中存储器的拼接

（3）存储分配算法

存储分配算法决定输入的程序和数据放到主存中的什么地方。常用的四种算法是：

①最佳适应法：选择最小的足够装入程序的可利用的自由区。对许多人来说，最佳适应看起来是最直观、吸引人的算法。

②首次适应算法：从主存低地址开始选择第一次足够装入程序的可利用的自由区。首次适应也具有直观吸力，此算法可以快速作出分配决定。

③最差适应算法：总是将程序装入主存中最大的自由区。这种方法吸引人的原因很简单：在自由区中放入程序后。剩下的自由区通常也很大，于是也能装下一较大的新程序。

④循环首次适应算法：不同于首次适应算法的是，从刚分配的自由区处开始寻找一个足够大的可利用的自由区。

4.交换与覆盖

上述四种方法都把用户作业完全连续地存放在一存储区域中，为了能在较小的主存空间中运行较大的作业，常采用交换技术和覆盖技术。

交换技术是指将作业不需要或暂时不需要的部分（进程）移到辅存，让出主存空间以调入需要的部分,交换到辅存的部分也可以再次被调入。实际上这是用辅存用缓冲，让用户程序在较小的存储空间空间中，通过不断地换出作业或进程而运行较大的作业。

覆盖技术是指让作业中不同时运行的子模块共同使用同一内存区域，这样，不必将程序完全装入主存即可运行。当运行中调用另一个模块时，再从辅存中调入这个模块而将原来已经运行完成的程序模块覆盖，即装入到同一个在存储区域内。对此，要求用户明确地描述作业中各个程序模块间的调用关系，这将加重用户负担。

3.4.2 虚存组织

虚拟存储通常涉及存储空间大于计算机系统主存中可利用的存储空间时的寻址能力问题。虚拟存储系统的特点是运行程序访问的地址不是主存中可以获得的，即运行进程访问的地址与主存可用的地址相脱离。 运行进程访问的地址不是主存中可以获得的，即运行进程访问地址与主存可用的地址相脱离。

运行进程访问的地址称为虚地址，主存中使用的地址称为实地址。一个可以访问的虚地址范围称为实地址空间。

虽然进程只访问虚地址，但它们实际上必须在实存中运行。有多种把虚地址与实地址联系起来的手段。当进程的虚拟地址空间连续，而实际地址空间不需要连续的性质。

1. 分段存储组织

在可变分区配方案中，主存中放置的程序常采用首次适应、最佳适应、最差适应或循环首次适应算法实现，但运行的程序需连续存放在一个分区中。

一个作业是由若干个具有逻辑意义的段（如主程序、子程序、数据段等）组成。在分段系统中，允许程序（作业）占据主存中若干分离的分区，如图 3.15 所示。

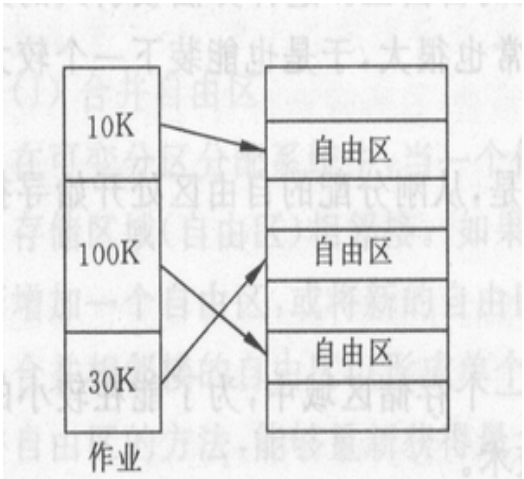


图 3.15 分段存储的不连续分配

每个分区存储一个程序分段。这样，每个作业需要几对界限地址，判定访问地址是否越界也比较困难。在分段存储系统中常常利用存储保护键实现存储保护。

分段系统中的虚地址是一个有序对（段号，段内位移）。系统为每一个作业建立一个段表，其内容包括段号与主存起始地址的对应关系、段长和状态等。状态指出这个段是否已调入主存，若已调入主存，则由主存起始地址指出这个段的位置，状态同时也指出这个段的访问权限。

分段系统的动态地址转换是这样进行的：进程运行时，其段表的首地址已在基本地址寄存器中。执行的指令访问虚存（s，d）（以指令或取消操作数）时，首先根据段号 s 查段表，若段已以调入主存，则得到该段的主存起始地址。然后与 d（段内偏移量）相加，得到实地址。如果该段尚未调入主存，则产生缺段中断，以便装入所需要的段。上述过程如图 3.16 所示。

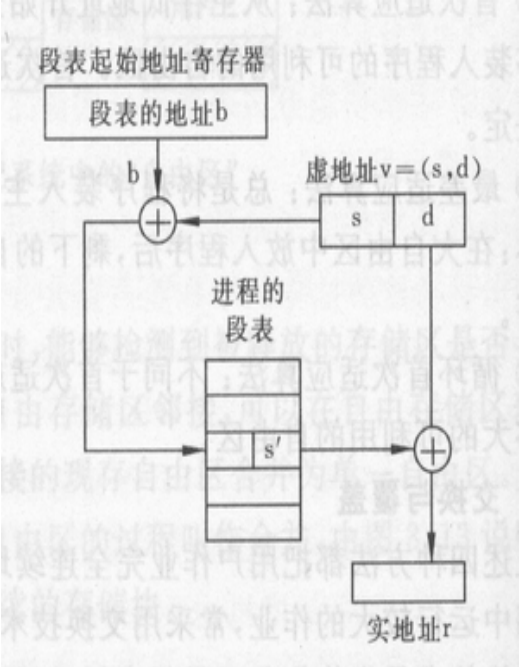


图 3.16 分段系统中的动态地址转换

2. 页式存储组织

页式存储组织与段式存储组织相似。但是，主存被分划成若干定长的页，页式系统中的虚地址是一个有序对（页号、段内页号、页内位移）。系统每一个进程建立一个页表，其内容包括进程的逻辑页号与物理页号的对应关系、状态等。 页式系统的动态地址转换是这样进行的：进程运行时，其页表的首地址已在系统的动态地址转换机构中的基本地址寄存器中，执行指令访问虚存地址（p，d）时，首先根据页号 P 查页表，由其状态可知，这个页是否已经调入主存。若已调入主存，则得到该页的主存位置，然后，与页内相对地址 d 组合，得到实地址；如果该页尚示调入主存，则产生缺页中断，以便装入所需的页。

3. 段页式存储组织

段页式存储组织综合了段式组织与页式组织的特点，主存被划分成定长的页。段页式系统中虚地址形式是（段号、段内页号、页内位移）。系统为每个进程建立一个表，为每个段建立一个页表。段页式系统的动态地址转换如图 3.17 所示。

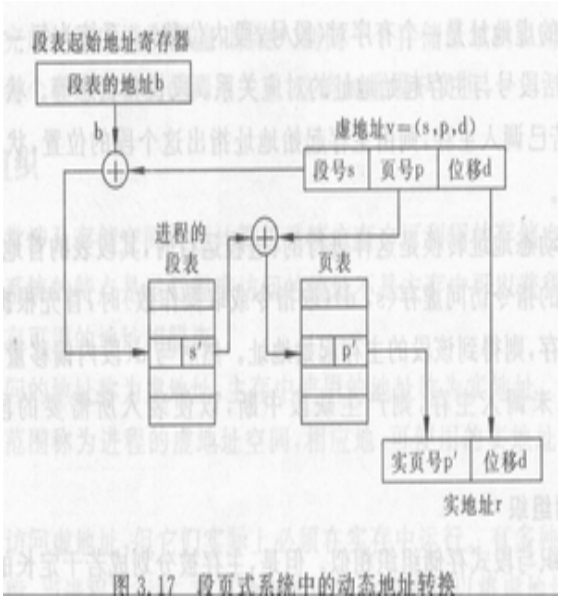


图 3.17 段页式系统中的动态地址转换

4. 联想存储器

一般，进程的段表和页表都相当大，通常存放在主存中。在采用页式存储组织与段式存储组织的系统中对段表与页表的访问需要一个完整的访问周期，而访问周期通常占据指令执行周期的主要部分，从而导致计算机系统的速度下降而不能容忍，因此必须使用更快的地址转换技术。

加快动态地址转换速度的一种方法是采用访问周期时间比主存周期时间快得多的联想存储器，带有联想存储吕的段页式存储组织的动态地址转换机构如图 3.18 所示。

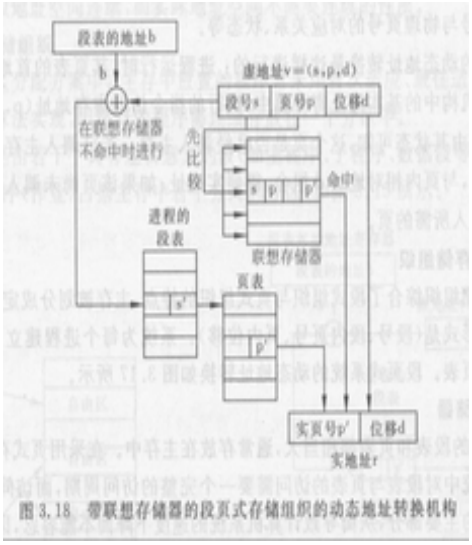


图 3.18 带联想存储器的段页式存储组织的动态地址转换机构

联想存储器采用平行比较技术按内容（段号、虚页号）得到相应的实页号（如果在联想存储器中）这种现象称为命中，则要先访问段表，再访问页表，得到相应的实页号与位移组合成实地址，再访问所需的存储单元。

3.4.3 虚存管理

在实存管理中讨论了调入、放置（放入分区）和交换（swapping）等问题，对于虚拟存储系统同样存在这些问题。

（1）调入策略：涉及何时将一页或一段从辅存调入主存。有两种算法：直到进程访问到某页或某段时，才把这个页或段调入主存，这称为请求调入方案；先行调入方案试图预测进程将要访问哪些页或段，在访问以前就先将这些页或段调入主存。

（2）放置策略：涉及将调入的页或段放在主存的什么地方。页式系统可以将其放置在任何一可利用的实例中，分段系统的作法则类似于可变分区分配系统。

（3）置换策略：涉及进程已用完该进程的可用主存空间时，选择淘汰哪些页段，腾出空间放置调入的页或段。在请求页式存储系统中，有若干淘汰算法（置换策略）：

①最优（OPT）算法：选择淘汰不再使用或最远的将来才使用的页，这是理想的算法，但难以实现，常用于淘汰算法的比较。

②随机（RAND）算法：随机地选择淘汰的页，开销小，但可能选中立即就要访问的页。

③先进先出（FIFO）算法：选择淘汰在主存驻留时间最长的页，似乎合理，但可能淘汰立即人使用页。另外，使用 FIFO 算法时，在未给予进程分配足够的页面时，有时会出现给予进程的页面数越多，缺页次数反而增加的异常现象，这称为 Belady 现象，例如，若某个进程访问页面的顺序（称页面访问序列）是 432143543215，当进程拥有 3 个实页面时，发生缺页率比拥有 4 个实页面时要小。FIFO 的异常现象如图 3.19 所示。



图 3.19 FIFO 的异常现象

④最近最少使用（LRU）算法：选择淘汰离当前时刻最近的一段内使用得最少的页。这个算法的主要出发点是，如果某页被访问了，则它可能马上就要被访问；反之，如何实现。一般需要提供硬件支持：如果每一页提供一个计数器，每当访问时即递增。另外一个实现方法是建立访问主存的栈，保证栈顶所指出的页总是刚访问的页，而栈底指出的页是 LRU 页。OPT 算法和 LRU 算法都不会发生 Bdlady 异常现象有若干个近似的 LRU 算法：

·使用次数最少（LFU）算法：选择淘汰到当前时刻为止，访问次数最少的一页。可为每一页设置一个计数器，并注意计数器需要复位，以免溢出无效。

·附加引入位算法：对每一页附加一个引用位以及若干附加位用于记数（如 8 位），操作系统定时将这些位右移一位，同时把引用位移入附加位的高位，这些附加位记录了访问相应页的历史。本算法将选择由这些位构成的数值中最小的那一页淘汰。在极端情况下，这些附加位组成的数值降为 0，而只留下引用位，这时相应的算法称为第二机会算法。

·第二次机会算法：这是基于 FIFO 的算法，然后，选择时先检查引用位，若尚未被置位则选中该页；反之，将引用位复位（给予该页第二次机会），再检查下一页。极端情况，蜕化成 FIFO 算法。

·最近不使用（NUR）算法：又称为增强的第二次机会算法。为每一页增加一位引用位和一位修改位，共有四类可能的情况：

(0，0)最近未被引用也未被修改，首先选择

（0，1）最近未使用，但被修改过

（1，0）最近使用而未修改（可能立即使用），再其次选择

（1，1）最近使用而且修改（可能立即使用），最后选择 存储管理策略的基础是局部性原理——进程往往会不均匀地高度局部化地访问主存。局部性分为时间局部性和空间局部性两类:时间局部性是指最近访问存储位置，很可能不久的将来还要访问，如循环、栈等；空间局部性是指存储访问有成组的倾向：当访问了某个位置后，很可能也要访问其附近的位置，如访问数组，代码顺序执行等。存储访问局部性最有意义的结果是，只要进程所需要的页面子集驻留在主存中，进程就可以有效地运行，根据局部性原理的特征性，Denning 阐述了程序性能的工作集理论。

简言之，工作集是进程频繁访问的页面的集合。工作集理论指出，为使进程有效地运行，它的页面工作集应驻留主存中。否则，由于进程频繁地从辅存请求页面，而出现称为“颠簸”（又称抖动）的过度的页面调度活动。此时，处理页面调度的时间超过了程序的执行时间。显然，此时 CPU 的有效利用率会急速下降。当一个进程陷入颠簸状态时，有的系统采用全局页面调度方法，即将其他进程拥有的主存页面调出主存供这个进程使用。然而，这样的处理又会导致被剥夺主存页面的进程，陷入颠簸状态。产生颠簸的根本原因是系统的负荷过大，通常采用处理

器的均衡调度，强制处于阻塞状态的进程让出主存；也可以采用带优先级的淘汰算法或局部淘汰算法,牺牲低优先级的进程，以控制颠簸的蔓延。 通常用两种等价的方法确定进程的工作集，一种是将工作集确定为在定长的页面访问序列（工作集窗口）中的页面集合，另一种是将工作集确定为在定长时间间隔中涉及到页面的集合，如图 3.20 所示。工作集的大小依赖于工作集窗口的大小，在进程执行时，工作集会发生变化。有时，当进程进入另一个完全不同的执行阶段时，工作集会出现显著的变化。不过在一个进程的执行过程中，工作集的大小处于稳定状态的时间基本上占绝大多数。

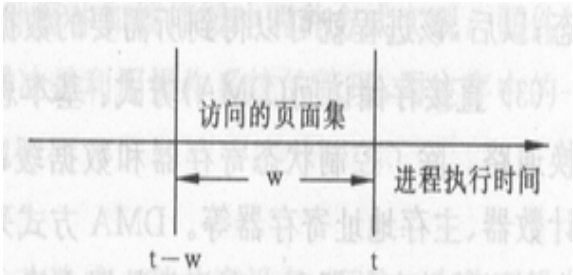


图 3.20 进程的页面工作集

另一种控制颠簸的技术是控制缺页率。操作系统规定缺页率的上下限，当一个进程的缺页率高于上限时，表明该进程需要更大的主存空间，则分配较多的主存页面给它，当进程的缺页率低于下限时，表明该进程占用的主存空间过大，可以适当地收回若干主存页面。

3.5 设备管理

在计算机系统中，除了处理器和主存之外，其他大部分硬设备称为外部设备,其中包括输入输出设备、辅存设备及终端设备等。这些设备种类繁多、特性各异，操作方式的差异很大，从而使操作系统的设备管理变得十分繁杂。

1. 设备的类型

80 年代以来，由于个人计算机、网络的发展，外部设备更趋多样化、复杂化和智能化。例如网卡、信真终端、虚拟终端、Windows 终端和 X-Windows 等终端等。

为简化设备管理程序，常对设备进行分类：

（1）按数据组织方式：字符设备（常对应于慢速设备）和块设备（常对应于快速设备）。

（2）近资源性质：独占设备（如终端、打印机），共享设备（如磁盘），虚拟设备（采用某种技术，将高速设备如磁盘虚拟化为多个“高速”的读卡机或打印机，以提高系统效率；这也称为 spooling 技术）。

2. 设备管理功能

设备管理是对计算机输入输出系统的管理，其主要任务是：

实现对外部设备的分配和回收，启动外部设备，控制输入输出设备与处理器与主存间交换数据，实现对磁盘的调度，处理设备的中断，实现虚拟设备等。

3. 数据传输控制方式

外部设备和主存之间常用的传输控制方式有四种：

（1）程序控制方式：处理启动数据传输，然后等待设备完成。

（2）中断方式：程序控制方式不能实现并发，中断方式可以实现并发，它的数据传输过程是这样的：进程启动数据传输（如读）后，该进程放弃处理器；当数据传输完成后，设备控制器产生中断请求，中断处理程序对数据传输工作处理以后，让相应进程成为就绪状态；以后，该进程就可以得到所需要的数据。

（3）直接存储访问（DMA）方式：基本思想是在外部设备和主存之间开辟直接的数据交换通路。除了控制状态寄存器和数据缓冲器之外，DMA 控制器中还包括传输字节计数器、主存地址寄存器等。DMA 方式采用窃取（或挪用）处理器的工作周期、控制总线来实现辅助存储器和主存间的数据交换。有的 DMA 方式也采用总线浮起方式传输大批量数据。

（4）通道方式：通道又称为输入输出处理机（IOP），可以独立完成系统产付的输入输出任务，通过执行自身的输入输出专用程序（称通道程序）进行主存和外设间的数据传输。主要有三类通道：

①字节多路通道：连接若干低速设备；

②选择通道：为一个高速设备服务；

③成组多路通道：同时可为多个高速设备服务。 通道的输入输出操作通过两种指令实现控制。一种是操作系统使用的专用输入输出指令，它指明进行输入输出的通道名、地址和设备，实现对通道的启动、复位、停止、查询等操作。对设备具有控制是由另一种指令即通道程序提供的，早期称为通道命令字（CCW），它完成对设备的具体控制。

4. 设备连接

现代计算机系统对外部设备的控制常分为三个层次，即通道、控制器和设备，如图 3.12 所示。控制制备的软件分为与设备有关的和与设备无关的两大类。前者与具体的设备类型有关，设备驱动程序就是与设备有关的；与设备无关的软件主要用于在用户提供统一的接口，包括处理的公共部分。需要处理的内容包括：为设备驱动程序提供统一的接口、设备命令、设备分配与回收、错误报告、块设备空间管理缓冲管理等等。

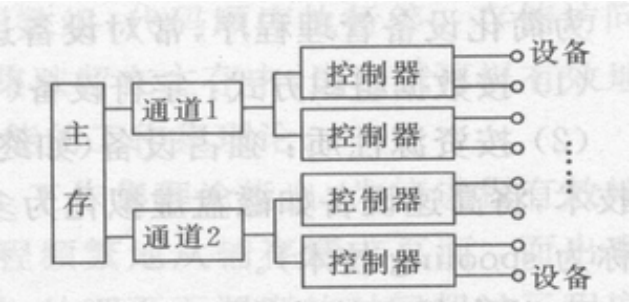


图 3.21 外部设备的连接

5. 组块与缓冲

通常，高速外设的输入输出操作以块为单位的，如磁盘块的大小是固定的。所谓块，又称为物理记录，是实际从设备读取或写到设备的信息单位。相应地，逻辑记录是从用户观点考察的一个信息单位。为了提高辅助存储器尤其是磁带的利用率，一般由若干逻辑记录组成一个物理记录，这称为组块技术。

缓冲是一种暂存技术。它利用某个存储设备，在数据的传输过程中进行暂时的存储。缓冲技术的引入，有效地改善了处理器与输入输出设备之间速度不匹配的情况，也减少了设备的中断请求次数。缓冲技术可以采用硬件缓冲和软件缓冲两种方式实现。硬件缓冲是利用专门的硬件寄存器作为缓冲区；软件缓冲区是利用操作系统的管理，用主存中的一个或多个区域作为缓冲区，进而可以形成缓冲池。

6. 虚拟设备

采用假脱机技术，可以将低速的独占设备改造成一种共享设备，而且一台物理设备可以对应若干台虚拟的同类设备。假脱机简称 spooling，它利用多道程序技术，采用一组程序或进程模拟一台输入输出处理机，脱机输出就是使用虚设备技术的典型例子。在现代计算机系统中，还可以用一台设备来模拟自身。如常见的多窗口技术，即在一个终端上开多个窗口，每个窗口可以独立地进行显示，以监视用户不同任务的执行情况，它是通过缩小显示区域，平铺或重叠显示来模拟多个显示器的。

3.5.1 设备的分配

设备分配的原则是按照特性和用户要求来确定的。设备分配的目的是既要充分发挥设备的使用效率，尽可能地让设备忙，又要避免形成死锁。另外，还要考虑到设备的独立性；即用户程序面对的是逻辑设备，分配程序要在系统把逻辑设备转换成物理设备之后，再根据要求的物理设备的状态进行分配。

设备分配方法有两种，即静态分配。静态分配方法是在用户作业开始执行之前，系统一次分配该作业所占有，直到该作业终止而撤消。显然，设备的使用效率较低。动态分配在进程运行过程中根据运行的需要而进行。当进程需要设备时，通

过系统调用命令申请设备，系统按照制定的策略分配相应的设备、控制器和通道给进程。一旦用完，便立即释放。动态分配方法用利于提高设备的利用率，但如果分配算法选用不当，将有可能产生死锁。

3.5.2 磁盘调度算法

设备的动态分配算法与进程调度相似，也基于一定的分配策略。常用的分配策略有先请求称分配、优先级高者先分配等。在多道程序系统中，低效率通常是由于磁盘类旋转设备使用不当而造成的。在操作系统中，对磁盘的访问要求来自多个方面，常常需要排队。这时，对众多的访问要求需根据磁盘调度算法按一定的次序响应；因此磁盘调度算法的性能会直接影响磁盘的工作效率，进而影响系统的性能。

访问磁盘的时间因子由三部分构成，它们是查找（查找磁道）时间、等待（旋转等待扇区）时间和数据传输时间，其中查找时间是决定因素。因此，磁盘调度算比考虑优化查找策略，需要时再优化旋转等待策略。

（1）先来先服务（FCFS）调度：按先来后到次序服务，未作优化。

（2）最短查找时间优先（SSTF）调度：FCFS 会引起读写头在盘面上的大范围移动，SSTF 查找距离磁头最短（也就是查找时间最短）的请求作为下一次服务的对象。SSTF 查找模式有高度局部化的倾向，会推迟一些请示的服务，甚至引起无限拖延（又称饥饿）。

（3）SCAN 调度：又称电梯算法。SCAN 算法是在磁头前进方向上的最短查找时间优先算法，它排除了磁头在盘面局部位置上的往复移动。SCAN 算法在很大程度上消除了 SSTF 算法的不公平性，但仍有利于对中间磁道的请求。

（4）N 步 SCAN 调度：这是对 SCAN 算法的另一种改良，是为磁头改变方向时已到达的请求服务的 SSTF 算法。（或为 n 个请求服务）

（5）C-SCAN 高度：这是对 SCAN 算法的另一种改良，是单向服务的 N 步 SCAN 算法，C-SCAN 算法彻底消除了对两端磁道请求的不公平。

3.5.3 RAID

RAID 即廉价的磁盘冗余阵列，是由若干个物理磁盘构成的一个阵列磁盘，数据分散存放在阵列中在的物理磁盘上，冗余的磁盘容量用来存放校验信息，用于磁盘失效时的数据恢复。RAID 有若干个等级（0-5 级）又可分成四类，如表 3.1 所示。

表 3.1 RAID 分类

分类	等级	说明	I/O 请求率 (read/write)	数据传输率 (read/write)	典型应用
条区	0	无冗余	大条区优	小条区优	对一般数据/高性能
镜像	1	数据镜像	良好/一般	一般	用于系统、重要文件
并行访问	2	Hamming 码纠错	差	优	不用大容量的 I/O
	3	位交叉奇偶校	差	优	请求，如 CAD
独立访问	4	块交叉奇偶校	优/一般	一般/差	不用高请求率
	5	分布块交叉	优/一般	一般/差	和数据查找

3.6 文件管理

在计算机系统中还有另一类资源是软资源。软资源包括各种系统程序、实用程序、各种应用领域的程序，也包括大量的文档材料。每一种资源本身都是具有一定逻辑意义的、相关信息的集合。在操作系统中糇们以文件的形式存储。在大多数应用中，文件是最重要的元素。每个应用总要使用和产生数据，一般，从文件中输入数据，其产生的数据也存入文件而长期贮存，供以后使用。计算机系统的重要作用之一是能快速处理大量信息，因此，数据的组织、存取和保护成为一个极重要的内容。文件系统是操作系统组、存取和保护数据的一个重要部分。

1. 文件的概念

文件是信息的一种组织形式，是存储在辅助存储器上的具有标识名的一组信息的集合。它可以是有格式的，也可是无格式的。

与文件相关的概念有：数据项、记录、文件和数据库。数据项是数据的基本单位，一个数据项有一个值，数据项可能是定长的也可能是可变长的。记录是相关数据项的集合，与数据项类似，记录的长度可能是定长的，也可能是变长度的。文件是记录的集合，它是一个实体，用户或应用程序可按名字访问。为安全起见，每一文件都有访问控制约束。数据库是相关数据的集合，数据元素间有直接的联系，这些联系是在设计时为若干个不同的应用而设计的。数据库本身可由若干文件组成。

2. 文件系统

操作系统的文件包括两个方面：一方面包括负责管理文件的一组系统软件，另一方面包括被管理的对象——文件。

文件系统的主要目标 提高存储器的利用率，接受用户的委托实施对文件的操作。为此要解决的主要问题是：管理辅助存储器，实现文件名字空间到辅存地址空间的转换，决定文件信息的存放位置、存放形式和存取权限，实现文件和目录的操作，提供文件共享能力和安全设施，提供友好的用户接口。

3. 文件类型

根据文件的性质和用途文件有多种分类方法：

（1）按文件的用途可以分为系统文件、库文件和用户文件等。

（2）按文件的信息流向可以分为输入文件、输出文件和输入输出文件等。

（3）根据文件的组织形式可以分为普通文件、目录文件和特殊文件等。特殊文件是 UNIX 系统采用的技术，是把所有的输入输出设备都视作文件（特殊文件）。特殊文件的使用形式是与普通文件相似的。

（4）按文件的安全性可分为只读者文件、读写文件、可执行文件和不保护文件等。

3.6.1.文件的结构和组织

文件的结构是指文件的组织形式，从用户观点所看到的文件形式，称为文件的逻辑结构;从实现观点考察文件在辅助存储上的存放方式,常称为文件的物理结构。

1. 文件的逻辑结构

文件的逻辑组织是为了方便用户的使用。一般，文件的逻辑结构可以分为两种：无结构的字符流文件和有结构的记录文件，后者也称为有格式文件。记录文件由记录组成，即文件内的住处划分成多个记录，以记录为单位组织和使用信息。

记录文件有堆文件、顺序文件、索引顺序文件、索引文件、直接文件和分区文件。

（1）堆（pile）文件：最简单的记录文件是堆文件。数据按先来后到的次序组织，每个记录所包含的数据项是自我标识的，数据项的长度可以明确指出或使用界定符区分。在堆文件中访问所需要的记录需穷尽搜索，这种文件组织不适合大多数应用。

（2）顺序文件：许多文件都是顺序文件。顺序文件的记录定长，记录中的数据项的类型长度与次序固定，一般还有一个可以惟一标识记录的数据项，称为键（key），记录按键值的约定次序组织。顺序文件常用于批处理应用，对于查询或更新某个记录的请求的处理性能不佳。

（3）索引顺序文件：索引顺序文件是基于键的约定次序组织的，而且维护键的索引和溢出区域。键的索引也可以是多级索引。索引顺序文件既适用于交互方式应用，也适用于批处理方式应用。

（4）索引文件：索引顺序文件是基于记录的一个键数据项组织的，而许多应用需按照其他数据项访问文件，为此，常采用索引文件方法，即对主文件中的记录本按需要的数据项（一个或几个）建索引,索引文件本身是顺序文件组织。

（5）直接文件：直接文件又称哈希（Hash）文件。可以运载记录在直接访问存储设备（DASD）上的物理地址直接（随机）访问。直接文件常用于需要高速成访问文件而且每次访问一条记录的应用中。

（6）分区文件：分区文件由两部分组成，一部分是顺序子文件区，另一部分阶段对这些子文件索引的索引区。分区文件常用于存储函数库、软件包等。

2. 文件的物理结构

文件的物理结构侧重于提高存储器的利用效率和降低存取时间。文件的存储设备通常划分为大小相同的物理块，物理块是分配和传输信息的基本单位。对于无格式的字符流文件，每一个物理块中存储同样长度的信息（除最后一块外）。对于有格式的记录文件，由于记录长度既可以是固定的，也可以是可变的，从而在存入辅助存储器时可能需要把若干个逻辑记录合并成一个块（称为组块）；相应地，在从文件存储器中读出时，需要把某个逻辑记录从一个物理块中提出来（称为解块）。

文件的物理结构是指文件在存储设备上的存储方法。文件的物理结构涉及文件存储设备的组块策略和文件分配策略，决定文件信息在存储设备上的存储位置，常用的文件分配策略有：

（1）顺序分配（连续分配）：这是最简单的分配方法。在文件建立时预先分配一个连续的物理块集，然后，按照逻辑文件中的信息（或记录）顺序，依次把信息（或记录）顺序存储到物理块中。这样，只需知道文件在文件存储设备上的起始位置和文件长度，就能进行存取。这种分配方法适合于顺序存取，在连续存取相邻信息时，存取速度快。其缺点是在文件建立时必须指定文件的信息长度，以后不能动态增长，一般不宜于需要经常修改的文件。如图 3.22 所示。

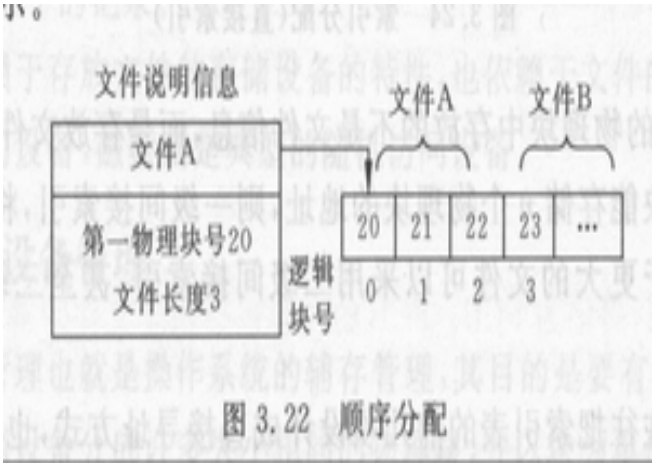


图 3.22 顺序分配

（2）链接分配（串联分配）：按单个物理块逐个进行分配。每个物理块中（一般是最后一个单元）设有一个指针，指向下一个物理块的地址，这样，所有的物理块都被链接起来，形成一个链接队列。在建立链接文件时，不需指定文件的长度，在文件的说明信息中，物理块间的指针就可以插入或删除一个信息块。链接

分配的优点是可以解决存储器的碎片问题，提高存储空间利用率。由链接文件只能按照队列中的链接指针顺序查找，因此搜索效率低，一般中适用于顺序访问，不适用于随机存取。如图 3.23 所示。

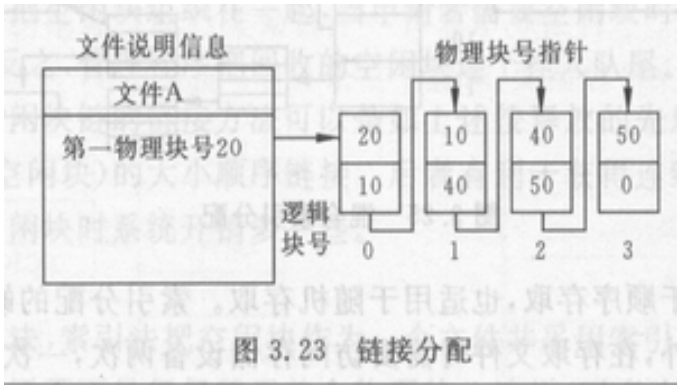


图 3.23 链接分配

（3）索引分配：索引分配是另一种对文件存储不连续分配的方法。采用索引分配方法的系统为每个文件建立一张索引表，索引表中的每一表项指出文件信息所在的逻辑块号和与之对应的物理块号，如图 3.24 所示。

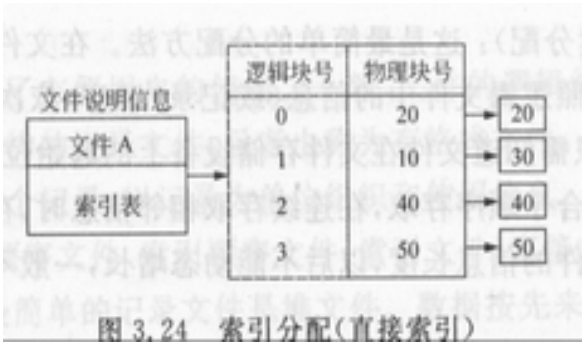


图 3.24 索引分配(直接索引)

索引分配既可以满足文件动态增长的要求，又可以方便而迅速地实现随机存取，这是因为索引表中含有全部逻辑号和物理块号之间的对应关系。对一些大的文件，当索引表的大小超过一个物理块时，会发生索引表的分配问题。一般采用多级（间接索引）技术，这时在由索引表指出的物理块中存放的不是文件信息,而是存放文件信息的物理块地址。这样，如果一个物理块能存储 n 个地址，则一级间接索引，将使可寻址的文件长度变成 n×n 块，对于更大的文件可以采用二级间接索引，甚至三级间接索引技术（如 UNIX 系统）

在实际系统中，往往把索引表的前几页设计成直接寻址方式，也就是说， 这几项物理块中存放的是文件信息；而索引表的剩余几页分别设计成一级间接、二级间接甚至三级间接寻址方式。这种混合方式，对小文件十分有效，也适应于大文件，如图 3.25 所示。

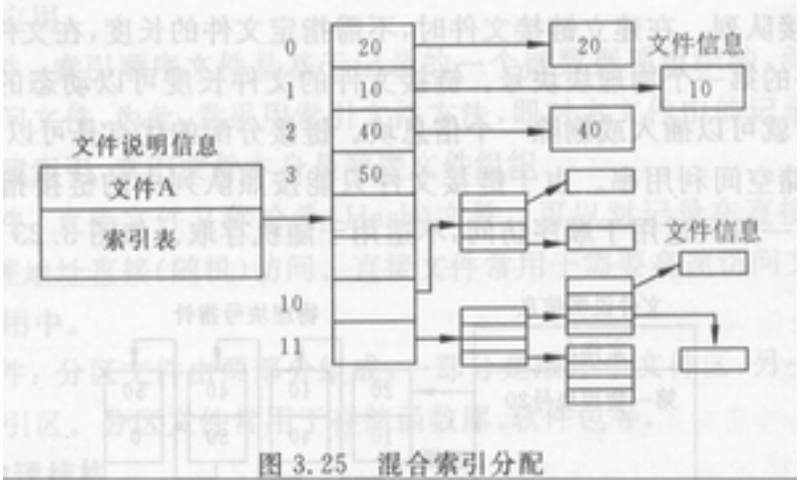


图 3.25 混合索引分配

索引分配既适用于顺序存取，也适用于随机存取。索引分配的缺点是索引表增加了存储空间的开销。另外，在存取文件时需要访问存储设备两次，一次是访问索引表，另一次是根据索引表提供的物理块号访问文件信息。为了提高效率，一种改进的方法是，在对某个文件进行操作之前，预先把索引表调入主存（如使用 Open 命令）；这样，文件的存取就能直接从在主存的索引表中确定相应的物理块号，从而中需要访问一次磁盘。

3.6.2 文件访问方法

用户通过对文件，访问（读写）来完成对文件的查找、修改、删除和添加等工作。常用的访问方法有两种：顺序访问和随机访问。

（1）顺序访问：按照文件的逻辑地址顺序访问。对无结构的字符流文件，顺序访问反映当前读写指针的变化。在访问完一段信息之后，读写指针自动加上这段信息的长度。

（2）随机访问：有两种方式：

• 根据记录的编号中的记录，或者根据读写命令把读写指针移到所需要的信息段开始处。

• 按键访问文件中的记录。

文件的访问依赖于存放文件的存储设备的特性，也依赖于文件的分配策略。例如，磁带是典型的顺序访问设备，磁盘则是典型的随机访问设备。

3.6.3 文件存储设备管理

文件存储设备管理也就是操作系统的辅存管理，其目的是要有效地进行存储空间的管理。由于文件存储设备分成许多大小相同的物理块，并以块为单位交换信息。

因此，文件存储设备的管理实质上对空闲块的组织和管理。 有三种不同的空闲块管理方法，它们是位图法、链接法和索引法：

1. 位图法

位图法使用一向量描述整个磁盘。向量的每一位表示一个物理块的状态，如 0 表示空闲块，而 1 表示该块已使用。位图法易于找到一个或连续几个空闲块，适合每一种文件分配方法；另外，位图法本身很小，易于全部放入主存。

2. 链接法

链接法使用链表把空闲块组织一起，当申请者需要空闲时块时，分配程序从链首开始摘取所需的空闲块；反之，管理程序把回收的空闲块逐个挂入队尾。这个方法适用于每一种文件分配方法。空闲块链的链接方法可以是如上述按释放的先后顺序链接，也可以按空闲块区（连续 n 个空闲块）的大小顺序连接。后者有利于获得连续的空闲块的请求，但在分配请求和回收空闲块时系统开销多一些。

3. 索引法

类似文件分配方法，索引法把空闲块作为一个文件并采用索引技术。为了更加有效，索引可以基于空闲块而不是仅仅基于单个物理块。这样，磁盘上每一个空闲块区都对应于索引表中的一个条目，这个方法能有效支持每一处文件分配方法。索引法和链接法配合的索引块链接方法如图 3.26 所示。

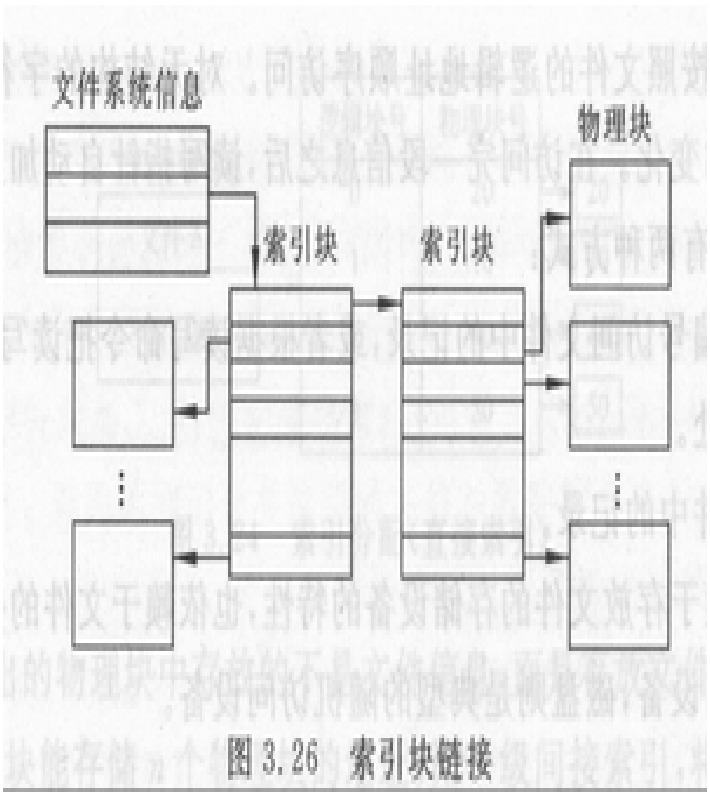


图 3.26 索引块链接

3.6.4 文件控制块和文件目录

1. 文件控制块

文件控制块是系统在管理文件时的必需的控制信息的数据结构，是文件存在的惟一标志，也称文件描述词，简记为 FCB。FCB 的内容包括相应文件的基本属性，大致可以分成四个部分：

（1）基本信息：如文件名、文件类型、文件组织等；

（2）保护信息：如口令、所有者名、保存期限、访问权限等；

（3）位置信息：如存储位置、文件长度等；

（4）使用信息：如时间信息、最迟使用者等。

2. 文件目录

文件控制块的集合称为文件目录，文件目录也被组织文件，常称为目录文件。

文件目录的组织与结构是文件管理的一个重要方面，也反映了文件系统的特色。一般，目录结构形式有一级目录结构、二级目录结构和多级目录结构。

（1）一级目录结构：文件系统中只有一个目录文件的目录结构称为一级目录结构。其中，每一表项是一个文件控制块，对应于一个文件。如图 3.27 所示。

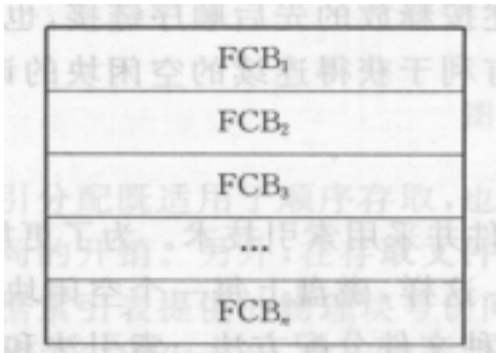


图 3.27 一级目录结构

一级目录结构易于实现，管理简单，但当系统中文件数增多时，查找时间较长，也会发生文件名重复（重名）的问题。

（2）二级目录结构：文件系统将目录分为两级，就形成了二级目录结构。这时，第一级目录称为主目录，主目录表项记录用户名及相应用户目录的存储位置。第二级是用户目录，以顺序表形式存放该用户文件的文件控制块，如图 3.28 所示。

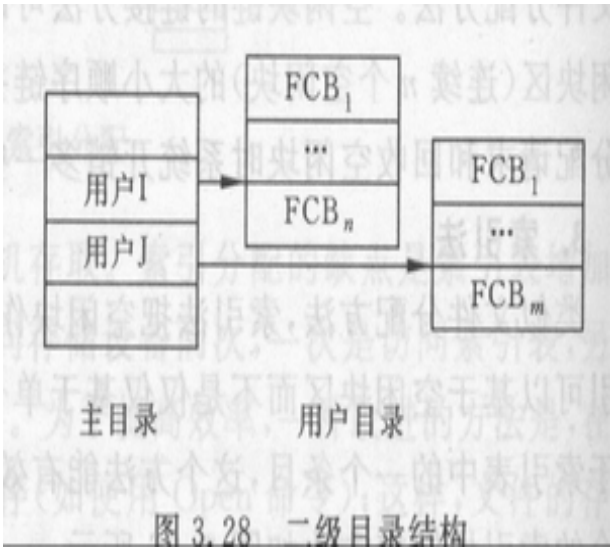


图 3.28 二级目录结构

二级目录虽然都组织成顺序表，由于都不会很长，查找时间快；而且各个用户只要保证自己的文件名称不重复就不会发生重名问题。对文件的共享和文件的保护也比一级目录结构好。

（3）多级目录结构：如果将二级目录的级数增加，就形成了多级目录。也称为树型目录结构。在多级目录结构中，常常将第一级目录作为系统目录，称为根目录（树的根节点）。目录树中的非叶节点指出目录文件，非目录文件一定由叶节点指出。叶节点也可能指出目录文件，即空目录。

在多级目录结构中，从根出发以到任何一个叶节点有且只须一条路径。该路径的全部节点名构成一个全路径名，又称绝对路径名。查找一个非目录文件必须使用它的全路径名。多级目录结构更加完善了文件结构的查找范围，更好地解决了文件的重名问题，增强了文件的共享和保护措施。多级目录结构如图 3.29 所示。

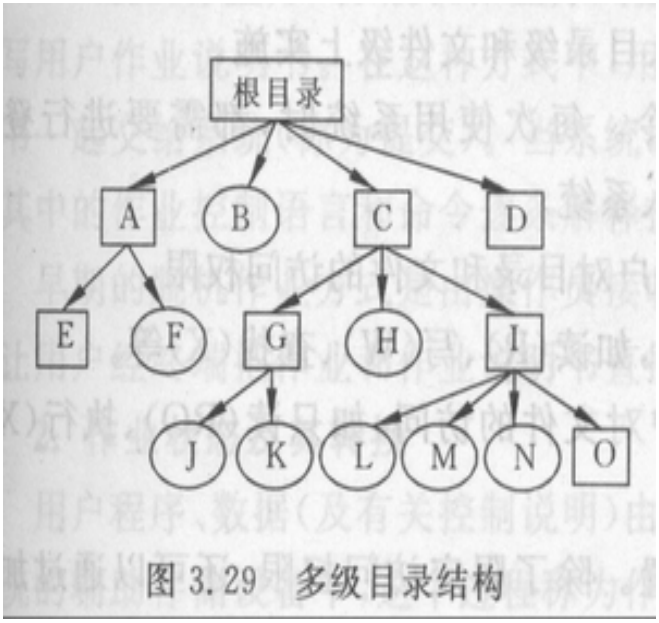


图 3.29 多级目录结构

图中用方框表示目录文件，用圆圈表示非目录文件（下同）。

有的系统允许一个非目录文件有两个全路径名，这称为文件的链接。这种作法有利于实现文件的共享，如图 3.30 中的文件 J 所示。

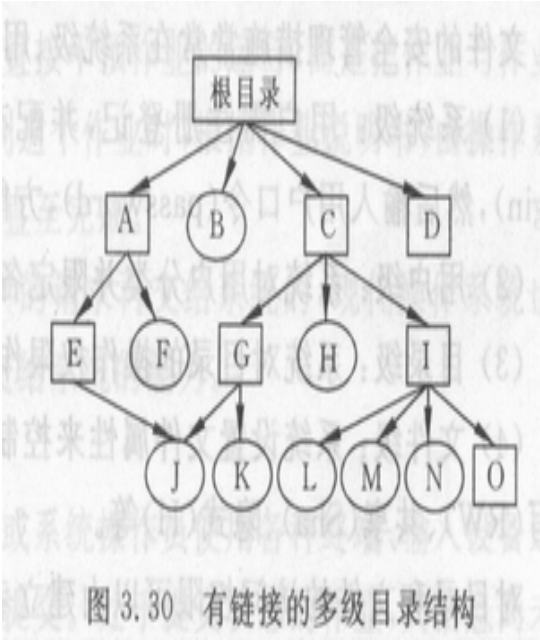


图 3.30 有链接的多级目录结构

3.6.5 文件的使用

1. 工作目录

也称当前目录。在多级目录结构的文件系统中，文件的全路径名可能较长，也会涉及多次磁盘访问。为了提高效率，操作系统提供设置工作目录的机制，每个用户都有自己的工作目录，任一目录节点都可以被设置为工作目录。一旦某个目录节点被设置成工作目录，相应的目录文件有关内容会被调入主存，这样，对以工作目录为根的子树内任一文件的查找时间将会缩短，从工作目录出发的文件路径名被称为文件的相对路径名，文件系统允许用户随时改变自己的工作目录（如命令 CD）。

2. 文件的使用

一般，文件系统提供一组命令专门用于文件、目录的管理。如目录管理、文件控制和文件存取等命令。

（1）目录管理命令：如建立目录、显示工作目录、改变目录、删除目录（一般只可删除空目录）等。

（2）文件控制命令：如建立文件、删除文件、打开文件、关闭文件、改文件名、改变文件属性等。

（3）文件存取命令：如读写文件、显示文件内容、自制文件等。

3. 文件共享和安全

文件共享和安全是指不同的用户使用同一文件。文件的安全是指文件的保密和保护，即限制未授权用户使用或破坏文件。可以采用文件的绝对路径名（或相对路径名）共享同一文件。有的系统允许文件的链接（如 UNIX 系统）。如图 3.30 所示，以目录 E 和子目录 G 为工作目录的两个用户可同时访问文件 J，称为静态文件共享。一般的文件系统，要求用户先打开文件，再对文件进行读写，不再使用时关闭文件。若两个用户可以同时打开文件，对文件进行存取，则称为动态文件共享。有时前者称为长期共享，后者称为短期共享。

文件的安全管理措施常常在系统级、用户级、目录级和文件级上实施。

（1）系统级：用户注册登记，产配有口令。每次使用系统时，都需要进行登录（login），然后输入用户口令（password），方能进入系统。

（2）用户级：系统对用户分类并限定各类用户对目录和文件的访问权限。

（3）目录级：系统对目录的操作权限作限定，如读（R）、写（W）、查找（X）等。

（4）文件级：系统设置文件属性来控制用户对文件的访问，如只读（PO）、执行（X）、读写（RW）、共享（Sha）、隐式（H）等。

对目录和文件的访问权限可以由建立者设置。除了限定访问权限，还可以通过加密等方式进行保护。

3.7 作业管理和用户界面

作业（job）是系统为完成一个用户的计算任务（或一次事务处理）所做工作的总和。例如，对于用户编的源程序，需经过对源程序编译、连接编辑或连接装入以及运行产生计算结果等步骤。这其中的每一个步骤。这其中的每一个步骤，常称为作业步，作业步的顺序执行即可完成作业。

操作系统中用来控制作业的进入、执行和撤消的一组程序称为作业管理程序，也可把作业步细化，通过进程的执行来实现这些控制功能。

用户的作业可以通过直接的方式，由用户自己按照作业步顺序操作，也可以通过间接的方式，由用户事先编写作业依次执行的说明，然后交给操作员，由系统按照说明依次处理。前者称为联机方式，后者称为脱机方式。

3.7.1 用户作业管理

1. 作业控制

（1）联机作业方式：联机作业方式通过直接输入作业控制命令的方式，提交用户作业，运行用户作业。用户作业的提交是通过终端依次键入操作命令或可执行程序名来提出运行请求。系统作出相应的处理，直至完成一个作业的计算要求。

命令的输入有两种方式。一种是顺序键入，用户每打入一条命令，操作系统接收命令，解释执行然后再等待用户找入下一条命令，直到作业完成为止。另一中是连续键入，用户打入一条命令之后，不等这条命令执行完成（让这条命令在后台运行），又可键入下一条命令，而连续键入的若干条命令形成了命令串，由操作系统自动地对这些命令逐个取出并解释执行，最后给出计算结果。

（2）脱机作业方式：

脱机作业方式通过作业控制语言（JCL，也称为作业控制命令），编写用户作业说明书。在这种方式中，用户不直接干预作业的运行，而是把作业与作业说明书一起交给系统（称为提交）。当系统调度到这个作业时，根据作业说明书，由操作系统对其中的作业控制语言和命令逐条解释执行，直至完成。

早期的脱机作业方式是由操作员接收用户的请求再交给系统的，现代操作系统也提供让用户经终端把作业和作业说明书直接提交给系统的能力。

2. 作业状态及其转换

用户程序、数据（及有关控制说明）由用户或系统操作员使用各种终端、输入设备送入系统的辅助存储设备中，这个过程称为作业的提交。处于提交状态的作业，其信息尚未全部进入系统。当作业的全部信息进入系统（如 spooling 系统中的输入井）时，该作业处于后备状态（又称为收容状态），操作系统为其建立作业控制块（JCB），JCB 中包含了作业的主要信息。作业调度程序从后备状态的作业中选取一个或若干个作业调入主存，建立进程并分配必需的资源，此时，这个（些）作业处于运行状态。当作业运行完成，该作业处于完成状态，系统将把该作业的结果输出，回收该作业占用的资源等簿记工作。作业的运行状态是由进程的状态描述的，如图 3.31 所示。

3. 三级调度

处理器调度问题就是处理器的分配问题。哪些作业的哪些进程可以参加竞争处理器呢？只有那些已取得必需的资源的里程才能竞争处理器，这些资源包括主存、外设及数据等。由于系统中会有很多进程，为提高主存利用率，操作系统将采用交换技术把主存中一些处于阻塞状态的进程换出主存，有时也把处于就绪状态的进程换出主存，以便让其他作业、进程可获得更多的主存。这样，处理器的分配有三个重要的调度级别：

（1）作业：作业调度又称高级调度。它决定哪一个（一些）后备状态作业可以竞争系统资源。作业一旦被选中，就建立相应的进程。

（2）均衡：均衡调度又称中级调度。它决定允许哪些进程竞争处理器，均衡调度通过使进程临时挂起和激活（或恢复）的方法对系统负载的波动作出反应，均衡系统的负载以实现较好的系统综合性能。

（3）进程：进程调度又称低级调度。它决定哪一个就绪状态的进程将分配到处理器，并把处理器实际分配给这个进程，即进行进程上下文切换以建立该进程的运行环境。

三级调度的示意如图 3.31 所示。

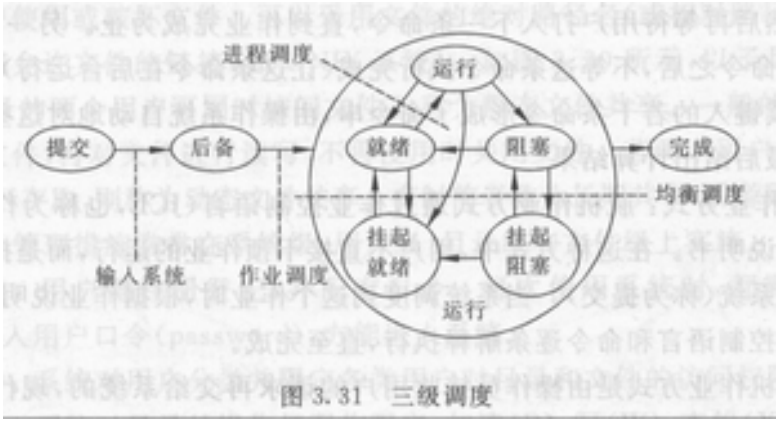


图 3.31 三级调度

3.7.2 作业调度

作业调度主要是从后备状态的作业中挑选一个（或一些）作业投入运行。如何选择呢？根据不同的目标，有不同的算法。一般，有若干调度目标：

1.调度目标

一般，调度目标主要有五点：响应时间快，周转时间或加权周转时间短，均衡的资源利用率，吞吐量大，系统反应时间短。这些目标是从不同的观点提出的，相互有冲突。通常，响应时间用于交互系统（如分时、实时系统）；周转时间或加权周转时间用于批处理系统，周转时间是指作业从提交到完成的时间间隔，加权周转时间是作业的周转时间与作业运行时间之比；系统反应时间是指作业从提交到获得处理器首次服务之间的时间间隔。

2.调度算法

作业调度算法有许多种，它们与进程调度相似，有的适宜于单道系统，有的适宜于多道系统。它们是：

（1）先来先服务（FCFS）：按作业到达的先后次序调度，它不利于短作业。

（2）短作业优先（SJF）：按作业的估计运行时间调度，估计运行时间短的作业优先调度。它不利于长作业，可能会使一个估计运行时间长的作业迟迟得不到服务。

（3）响应比高者优先（HRN）：综合考虑上述两方面的因素，即考虑作业的估计运行时间。又考虑作业的等待时间，响应比是：

HRN=（估计运行时间+等待时间）/估计运行时间

（4）优先级调度：根据作业的优先级别，优先级高者先高度。

3.7.3 用户界面

用户界面也称为用户接口，其含义有两种，一种是指用户与操作系统交互的途径和通道，另一种是指这种交互环境的控制方式，即操作系统。

1.操作系统的接口

操作系统的接口又可分成命令接口和程序接口。

（1）命令接口：命令接口包含键盘命令和作业控制命令；

（2）程序接口：程序接口又称为编程接口、系统调用。程序经编程接口请求系统服务，即通过系统调用，程序与操作系统进行通信。

2.操作环境

操作环境支持命令接口和程序接口，提供友好、易用的操作平台。操作系统的交互界面已以从早期的命令驱动方式，发展到菜单驱动方式、图符驱动方式和视窗操作环境。

3.8 其他管理

3.8.1 死锁问题

进程管理是操作系统的核心，在进程管理的实现中，如果设计不妥，会出现一种尴尬的局面——死锁。如果一个进程正在等待一个不可能发生的事件，则称该进程处于死锁状态。系统发生死锁，是指一个或多个进程处于死状态。

1.死锁条件

产生死锁的主要原因是供共享的系统资源不足，资源分配策略和进程的推进顺序不当。系统资源即可能是可重用永久性资源，也可能是消耗性的临时资源。关于可重用资源产生死锁存在四个必要条件，它们是：

（1）互斥条件：又称独占条件，即一个资源每次只能被一个进程使用。

（2）保持和等待条件：又称部分分配条件，即一个进程已获得一些资源，又因请求其他资源而被阻塞等待。

（3）不剥夺条件：进程已获得的资源，不经进程释放，不能剥夺这个资源。

（4）环路等待条件：若干进程形成一个环形链，其中每个进程占有链中下一个进程所申请的一个或多个资源。

2.解决死锁的策略

虽然，人们不希望发生死锁。处于死锁状态的进程不能继续运行又占有了系统资源，从而会阻碍其他进程的运行。对待死锁，一般，应考虑下列四个问题：

（1）死锁的预防：不让任一必要条件发生就可以预防死锁。如要求用户一起申请所需的全部资源，这破坏了保持和等待条件；又如可以把资源分成若干层次，记为 L1，L2，…Lm，规定一进程获得 LI 层次以后，只可再请求获得 Lj（j＞i）层资源，这个方法破坏了环路等待条件。

（2）死锁的避免：死锁预防措施会降低系统效率，资源利用率也不好。死锁的避免不对用户进程的推进加以限制，在进程申请资源时先判断这次分配安全否（安全意味着全部进程都可完成），安全才实施分配，典型的算法是银行家算法。

（3）死锁的检测：死锁的避免算法增加了系统的开销，死锁的检测采用资源请求分配图的化简单判断是否发生了不安全状态。资源请求分配图是一种有向图，表示进程与资源之间的关系。图中进程用圆圈表示，资源用方框表示。由资源指向进程的有向过表示该资源已分配给此进程（称为分配边），由进程指向资源的有向边表示进程正申请此资源(称为请求边)，图 3.32 是一个例子。死锁的检测是在需要的时刻执行的，当发现系统处于不安全状态时，即执行死锁的解除策略。

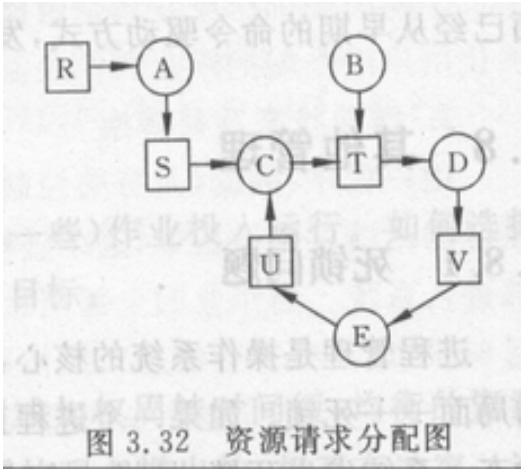


图 3.32 资源请求分配图

（4）死锁的解除：解除死锁的基本方法是剥夺。一种方法是把资源从一些进程处剥夺分给别的进程，被剥夺资源的进程则需回退到请求资源处重新等待执行；别一种方法是终止一进程，剥夺其全部资源，以后再重新运行被终止进程。

3.8.2 多处理器系统与线程

近年来,随着微电子技术的发展,使计算机系统由单处理系统发展到多处理系统。其中后者又可按是否有共享主存分成紧密耦合的多处理器系统和由独立计算机系统构成的网络系统。网络系统又可分局域网、城域网、广域网等。

1.紧密耦合多处理器系统的分类

紧密耦合多处理系统简称多重处理系统。在多重处理系统中，进程可在各处处理器间透明迁移，因此，由于进程的并发执行，进程上下文切换等系统开销使得多重处理系统的复杂度大大增加。多重处理系统的主要目标是为了提高系统的处理能力，也为了提高的可靠性（当某个处理器发生故障时，系统仍能正常工作——降低使用）。因此，多重处理系统的操作系统除了单处理器操作系统的功能以外，还应提供处理器的负载平衡、处理器发生故障后的结构重组等。 一般，多处理系统的操作系统可以分为三类：

（1）主从式：在主从式操作系统中，主处理机上运行操作系统，控制其他从处理机的状态，为从处理机分配任务（处理机如同一般资源）。显然，一旦主处理机发生故障，系统会瘫痪。

（2）分离执行：在分离执行系统中，各处理机运行各自的管理程序（或监督程序），为进程提供服务。一般，管理程序的代码是可以重入的，各处理机上有管理程序的副本。各处理独立工作，冲突少，但不容易实现负载平衡。

（3）移动执行：在移动执行系统中，管理程序根据需要从一个处理机迁移到另一处理机上。这种方式易于实现负载平衡,但对系统数据结构的访问需注意互斥。

多重处理系统中除了互斥锁还提供旋转锁。

2.对称多处理系统与线程

对称多处理系统（简记为 SMP 系统）是近年来流行的计算机系统。它由若干同构甚至相同的处理器构成一个系统。如 Solaris 和 WindowsNT 等就支持 SMP 系统。为了发挥多个处理器的作用，操作系统提供了线型（thread）机制，线程也称为轻进程（light-weight process）。

在多线程系统中，一个进程可以由一个或多个线程构成。进程是资源分配的基本单位，也是被保护的基本单位。一个进程对应于一个保存进程映像的虚地址空间，每一线程可以独立运行，具有各自的状态（运行、就绪等），有自己的栈、局部变量，有上下文转换，一个进程的线程共享这个进程的地址空间。一个进程中的线程间的上下文转换的系统开销比进程的上下文转换开销小。在 SMP 系统中，使用多线程完成用户的计算任务可充分发挥多处理器的能力，更有利于前后台工作、异步处理等应用，可以加速程序的执行。有多种方法实现多线程系统，一种方法是核心级线程，另一种方法是用户级线程，也可以把两者组合，其示意如图 3.33

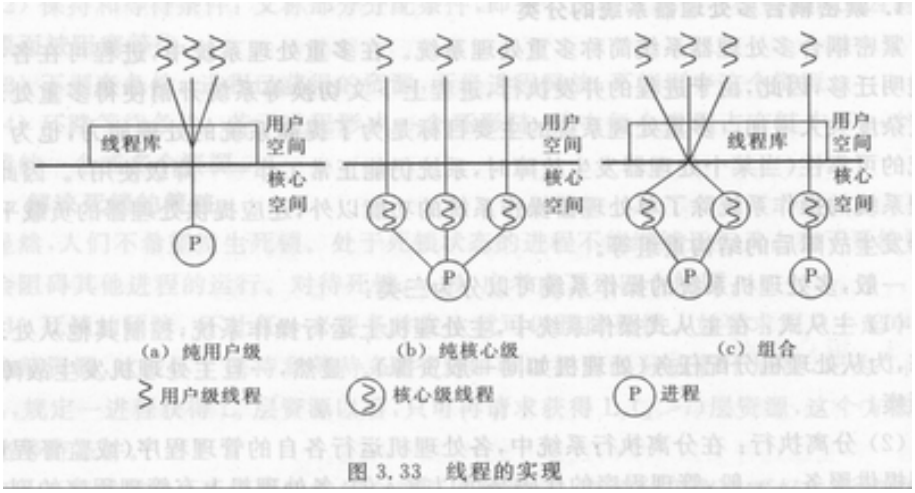


图 3.33 线程的实现

3.8.3 操作系统的结构

操作系统是典型的系统软件，是大型的并行工作程序组，不仅程序数量巨大，而且各部分之间的逻辑关系复杂。所以，研究操作系统的结构对缩短操作系统的开发周期，保证正确性、可靠性，以及易读、易维护、易移植十分重要的意义。

1.无序结构法

又称整体结构或模块组合结构。它以大型表格和队列为中心，表格和队列以全程数据结构形式出现，操作系统的各部分程序围绕着表格运行，只有主程序、子程序的概念（亦即整个系统是一个程序），调用、转移随便。具有这种结构的系统中有用户并行的概念，操作系统的内部不能并行。显然，这种系统结构紧凑，运

行效率高，但各部分之间联系复杂，安全性较难保证，增、删、改困难，适用地较小型的系统。这种操作统常称为面向过程的操作系统。

2.层次结构法

层次结构法是把一个大型复杂的操作系统分解成若干个单向依赖的层次，由各层的正确性保证操作系统的正确性。层次结构不仅结构清晰，而且有利于系统功能的增加中删改。

3.面向对象法

面向对象的操作系统基于面向对象程序设计的概念，采用了各种不同的对象技术。面向对象的基本概念包括：对象、类、封装、继承、消息、多态和动态联编等。

在计算机系统中的对象是操作系统管理的信息的资源的抽象，是一种抽象数据类型，即数据表示，它们是数据结构和在这些数据结构上的操作。数据结构代表数据文件、可执行模块、文件目录、硬件资源以及相应的控制软件。对象拥有自己的状态或存储空间，状态或存储内容是由自带的若干操作改变的。这些操作由其他对象发出的相应信息而激发，因此操作系统中的对象作是受保护的.可以把对象作为系统中的最小单位,利用对象、对对象的操作加上保护组成操作系统，这就是面向对象的操作系统。

如 WindowsNT 中有执行体对象（进程、线程、文件、令牌等）、内存对象（时钟、事件、信号等）。

4.微内核是把系统的公共部分抽象出来，形成一个底核心，提供最基本的服务，其他功能以服务器形式建立在微内核之上。它具有良好的模块化和结构化特征，模块之间和上下层之间经消息通信。建立在微内核上的服务器可以根据不同的需要构造，从而形成不同的操作系统功能，如 WindowsNT 操作系统采用类似于卡内基·梅隆大学开发的 Mach 操作系统的微内核技术，把机制从策略中分离出来，微内核完全避免制定策略。在上层采略而支持不同的系统标准，如 POSIX 标准、OS/2 系统等。

3.9 实例

3.9.1 UNIX 系统

1. 系统结构

图 3.34 是 UNIX 系统结构的示意图。方框中是 UNIX 系统核心组成的示意图。UNIX 是一个分时操作系统。它利用最内层硬件提供基本服务，向外层提供全部应用程

序所需要的服务。应用程序组可以构成应用子系统，如 UNIX 系统的源代码控制系统（SCCS）、图形系统（X-Window，Motit）等等。

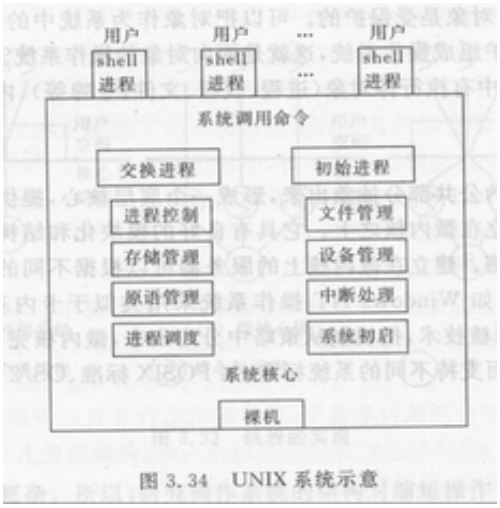


图 3.34 UNIX 系统示意

2. 文件系统

UNIX 中的文件是字符流文件，在物理上组织为索引文件。文件系统被组织成层次结构，有一个根节点，记住"/"。每一个非叶节点都是目录文件，叶节点既可是目录文件，也可是非目录文件或设备特殊文件。文件名由全路径名标识，允许进行文件连接（在符合权限前提下），因此，同一个文件可以有多个全路径名。这了提高文件的存取速度，UNIX 提供了工作目录的概念，用户可以指定或改变工作目录，这时使用文件相对路径名。

进程通过系统调用要求文件系统提供服务。例如：Open--打开一个文件，准备读、写；Close--关闭文件；Write--写文件；Read--读文件；Stat--查询一个文件的属性；Chown--改变文件所有者；Chmod--改变文件存取许可权；Creat--创建一个普通文件；Mkdir--创建一个目录文件；Cd--改变当前目录；Link--建立连接；Unlink--删除文件连接；等等。

3.进程管理与存储管理

UNIX 系统中的进程控制子系统负责进程同步、进程间通信、存储管理及进程调度。

用于控制进程的系统调用有：Fork--创建一个子进程；Exec--改变执行程序的映像；Exit--结束一个进程的执行；Wait--暂停进程的执行，用于进程之间的同步，例如父进程等待子进程执行结束；Signal--控制进程对特别事件的响应；Kill--发送软中断信号；Msgsnd--发送信息；Msgrcv--接收消息；等等；

UNIX 系统中的进程调度采用动态优先数的算法，当进程因等待资源而自愿放弃 CPU，或者因为它最近一次运行时间超过一个时间片而被剥夺 CPU 时，调度程序挑选优先级最高的进程投入运行。其优先数计算方法为：

使用 CPU 的时间/系数+基础优先数+nice 值

nice 值可由用户设定。优先数越小，优先级别越高。UNIX 进程间通信的方式不止一种形式，如事件的软中断信号、进程间消息同步传输、缓冲区交换等。

UNIX 采用段页式管理，分配单位是块（如每块 8K 字节）。UNIX 采用了 LRU 作为虚存中的淘汰算法。同时 UNIX 还使用交换技术，无论在什么时刻，只要系统没有足够的主存供进程使用，就在主存与辅存之间实施进程技术，可以有效地避免颠簸现象的出现。而且，它还采用了一些技巧来改善虚存的性能，如偷页（提前确定下一个将被淘汰的页面）、自由区域（将淘汰页暂时放入主存的自由区域--实际上部分地体现了全局淘汰）等等。

4.缓冲机制和设备驱动

UNIX 文件系统用缓冲技术来调节系统与磁盘之间的数据流。它设置了一个缓冲池（由若干个缓冲区组成），块设备驱动程序只与缓冲池交换数据。而且 UNIX 还采用"提前读"、"迟延写"的技术。即在可能的情况下把用户即将使用的文件内容（例如字符流文件当前读写位置以后的若干块）提前读到缓冲区；将用户写入文件的内容暂放在缓冲区，等到了有相当数量（或缓冲池空间紧张）时再写回磁盘。这样，不必出现用户每次使用文件（读/写）一定导致启动磁盘的现象，从而提高了效率。

同时，文件系统还提供无缓冲机制的设备访问功能。

5.中断处理

硬件控制负责接收中断，负责与机器的通信。设备可以在一个进程正在执行时发出中断，在 UNIX 中，中断是由核心中的特定函数来处理的，在当前进程的运行是调用这些函数处理相应的中断。

6.Shell

UNIX 系统中的命令解释程序称作 shell。

命令行的一般形式是：命令名[参数，参数，…]。命令行要以带若干参数，也可以不带参数。一个命令名对应于一个可执行文件。Shell 在收到命令以后，查找相应文件，将其调入主存并且执行，命令行中参数在相应程序执行时加以使用。

Shell 向用户提供了重定向操作与管道操作，以便用户可以将若干程序段组合起来，构成一个较大、较复杂的程序。为此目的，shell 提供了一般程序设计语言的语言成分，如 for,case,if-then,while 等。这样，利用 shell 语言可以编制出功能很强的 shell 程序（常称 shell 文件），将程序段组合起来。

UNIX 系统中有三个标准文件：标准输入文件（0 号文件）、标准输出文件（1 号文件）和标准错误文件（2 号文件），分别对应于键盘、显示屏和显示屏。而且，这些文件可以独立地被重新定向。例如，命令：

Ls

把工作目录中的内容显示在标准输出文件上（即屏幕）。但命令：

ls>outfile

则将 ls 的输出重定向到当前目录下的文件 outfile 上(若不存在,则创建之)."＞"是标准输出重定向符号.标准输入重定向的符号为"＜",标准错误定向的符号为"2＞"。另外，"＞＞"表示标准输出重定向且拼接。

管道是 UNIX 系统的一个重要特色，它允许在生产者进程与消费者进程之间传递数据流。管道机制建立一个临时的"哑"文件（管道文件），把生产进程的标准输出重定向到管道文件上，并启动消费者进程，将它的标准输入连接起来。管道的记号为"|"。

使用管道机制可以简化命令行的写法，如三个连续的命令行：

ls>file1

sort<filel>file2

pr<file2

可以表示为

ls|sort|pr

此外，UNIX 系统还提供后台命令，例如

cc prog.c &

表示用户可以不必等到对 prog.c 编译结束就可键入新的命令，shell 可以响应并处理新的命令，从而向用户提供了多个任务的并行。

7. 系统调用

UNIX 系统提供了近 60 个系统调用，允许用户在自己的程序中调用它们。系统调用通过 trap 指令执行用户程序切换到执行系统中相应的程序。

3.9.2 Windows NT 系统

Windows NT 是 Microsoft 1993 年推出的采用 90 年代技术的操作系统，适用于高档工作训平台、局域网服务器或者主干计算机。Windows NT 支持对称多处理结构，支持多线程并行，采用 16 位标准字符集的单一代码方式（Unicode 代码），支持国际字符集，提供性能良好的文件系统。因此，它一问世就受到了人们的新睐，发展的势头相当迅猛，已经成为在中小型计算机上颇具影响力的操作系统。Windows Nt 采用 20 世纪 90 年代操作系统技术（即微内核技术），在体系结构上采用客户/服务器模式。在概念上、风格上与传统的操作系统迥异，它集成了网络技术，可以方便地支持在网络环境下运行的要求。

1. 系统结构

图 3.35 是 Windows NT 系统结构的示意图。它可以分为两大部分：Windows NT 执行体与保护子系统（服务器部分）。前者在核心态下运行，后者在用户态下运行。Windows NT 有两类保护子系统:环境子系统和集成子系统.每个环境子系统是一个用户态的服务器进程,它可以仿真一个特定子系统,提供应用编程接口(API),因此,WindowsNT 可以向用户提供多种操作系统的服务器模式.WindowsNT 最重要的环境子系统是 Win32 系统，它向用户提供 32 位 Windows 的应用编程接口，而且还提供图形用户接口，并控制所有用户的输入/输出。此外 Windows NT 还提供了 POSIX 环境子系统、OS/2 环境子系统、MS-DOS 环境子系统和 16 位 Windows 环境子系统。这样，UNIX 用户、OS/2 用户、DOS 用户与 Windows 用户均可以以自己习惯的方式在 Windows NT 的支持下工作。集成子系统是完成操作系统重要功能的服务器，例如，安全子系统与网络子系统等。

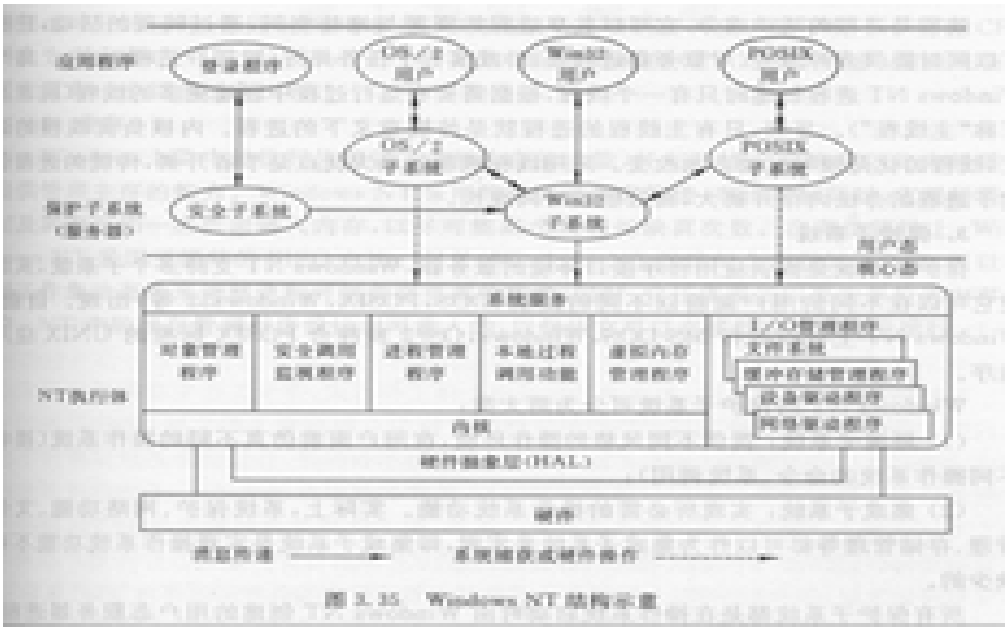


图 3.35 Windows NT 结构示意

Windows NT 的执行体在核心态下运行，除了用户接口以外，NT 执行体本身是一个完整的操作系统，提供一般传统操作系统支持的所有功能。执行体包括：对象管理程序、安全引用监视器、进程管理、本地过程调用、虚存管理、内核（中断、异常和线程调度）、I/O 系统（I/O 管理、文件系统、设备驱动、网络服务）以及硬件抽象等。

2. 进程与线程

在 Windows NT 执行体中，进程是由对象管理器创建和删除的对象。每个进程创建时只有一个线程。根据需要，进程可以创建更多的线程。大多数进程是用户态进程，用户态进程通过系统调用访问操作系统。当核心态完成服务，在将控制权交不还用户程序前再将线程切换回用户态。应用程序在用户态下运行， Windows NT 保护子系统也是如此。后者是提供重要的操作系统功能的用户态服务进程。

多线程进程实现的并行避免了进程间并行的缺点：创建线程的开销比创建进程要小，同一进程的线程共享进程的地址空间，所以线程切换（处理器调度）比进程切换快。NT 内核采用基于优先级的方案选定线程执行的次序。高优先级线程先于优先级线程执行，内核周期性地改变线程的优先级，以确保所有线程均能执行。

线程是进程的活动成分，它可以共享进程的资源与地址空间，通过线程的活动，进程可以提供多种服务（对服务器里程而言）或实行子任务并行（对用户进程而言）。每个 Windows NT 进程创建时只有一个线程，根据需要在运行过程中创建更多的线程（前者亦可称"主线程"）。显然，只有主线程的进程就是传统意义下的进程。内核负责线程的高度，线程的优先级可以动态地改变。采用线程机制的最大优点是节省开销，传统的进程创建子进程的办法内存开销大，而且创建时间也长。

3. 保护子系统

保护子系统是提供应用程序接口环境的服务器，Windows NT 支持多个子系统，从而使它可以在不同的用户面前以不同的面貌(DOS,POSIX,Windows32 等)出现。目前，Windows NT 上可以运行 MS-DOS,Windows,OS/2 和 POSIX 标准的 UNIX 就用程序。

Windows NT 的保护子系统分成两大类：

（1） 环境子系统：提供不同风格的操作环境，在用户面前仿真不同的操作系统（接收不同的操作系统的命令、系统调用）；

（2） 集成子系统：实现所必需的操作系统功能。实际上，系统保护、网络功能、文件管理、存储管理等都可以作为集成子系统来实现，即集成都成子系统是实现操作系统功能不可缺少的。

所有保护子系统都是在操作系统启动时由 Windows NT 创建的用户态服务器进程。Windows NT 的网络服务器可以在用户态下运行，也可以在核心态下运行（取决于如何设计）。它们在执行体的支持下工作，通过调用核心状态下运行的系统服务（它们提供基本服务功能），以自己的形式与风格向用户提供服务。

子系统之间通过消息传递相互通信。每个子系统都设置了一个端口用于通信。当两个进程处于同一计算机时，使用本地过程调用（RPC）机制。一般地，用户进程与环境子系统之间也用同样的方式进行通信。Win32 子系统是 Windows NT 中

最基本的子系统，它由窗口管理程序、图形设备接口、操作系统功能块、控制台（提供文字窗口支持）以及图形设备驱动程序所组成。它向用户提供 32 位 Windows 功能，并控制所有用户的输入与输出。

其他几个环境子系统分别提供了不同的操作系统（DOS，OS/2，UNIX）的应用编程接口。

4. 内核

内核完成的主要任务包括：线程调度、中断接收及控制、处理低层多处理器之间的同步以及系统恢复。内核常驻内存，在核心态下运行，处理所有与 CPU 有关操作。线程同样也可分为运行、等待、就绪状态，以优先级为基础进行调度。除了将一个物理 CPU 转化为多个虚 CPU 之外，Windows NT 的内核还要处理多 CPU 之间的同步，保证在临界区内只有一个处理机在执行。

5. 虚存

Windows NT 中的执行体提供了虚拟内存的功能，任何环境子系统都以此为基础来提供管理主存的能力。Windows NT 采用按需页面调度算法，当出现缺页时，它把所需页面及其附近的一些页面调入内存，以尽可能减少线程的缺页次数。在淘汰算法上，Windows NT 采用了简单的 FIFO 方法。另外,Windows NT 跟踪每个进程的工作集，可以根据工作集动态地用来进程可用内存页面的数量；同时，它还实现了以页为单位的内存保护。NT 中的虚存管理程序设计成可重入的，以保证它可以在多处理机系统中运行。

软件工程

第一章 软件工程概述

一、复习要求

1. 了解软件概念、特点及分类方法。

2. 了解软件发展及软件危机的起因。

3. 了解软件工程过程及软件生存期的概念。

4. 了解软件工程的概念及其要素。

5. 了解软件工程的基本目标和原则。

二、内容提要

1. 软件的概念、特点

软件是计算机系统中与硬件相互依存的另一部分，它是包括程序，数据及其相关文档的完整集合。其中，程序是按事先设计的功能和性能要求执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发，维护和使用有关的图文材料。

软件的特点是：

(1) 软件是一种逻辑实体，而不是具体的物理实体。因而它具有抽象性。

(2) 软件的生产与硬件不同，它没有明显的制造过程。对软件的质量控制，必须着重在软件开发方面下功夫。

(3) 在软件的运行和使用期间，没有硬件那样的机械磨损，老化问题。任何机械、电子设备在运行和使用中，其失效率大都遵循如图 1.1(a)所示的U型曲线（即浴盆曲线）。而软件的情况与此不同，因为它不存在磨损和老化问题。然而它存在退化问题，必须要多次修改（维护）软件，如图 1.1(b)所示。

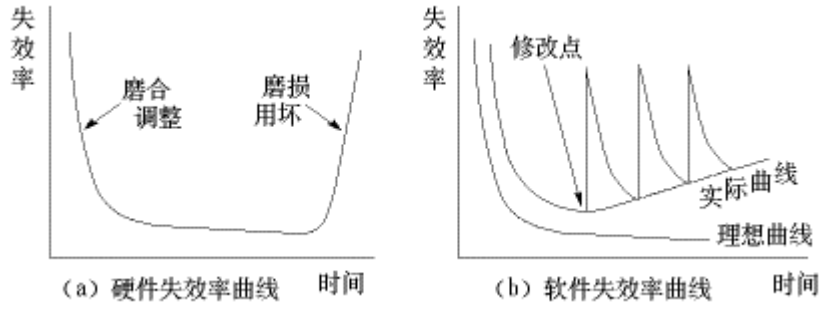


图 1.1 失效率曲线

(4) 软件的开发和运行常常受到计算机系统的限制，对计算机系统有着不同程度的依赖性。为了解除这种依赖性，在软件开发中提出了软件移植的问题。

(5) 软件的开发至今尚未完全摆脱手工艺的开发方式。

(6) 软件本身是复杂的。软件的复杂性可能来自它所反映的实际问题的复杂性，也可能来自程序逻辑结构的复杂性。

(7) 软件成本相当昂贵。软件的研制工作需要投入大量的、复杂的、高强度的脑力劳动，它的成本是比较高的。

(8) 相当多的软件工作涉及到社会因素。许多软件的开发和运行涉及机构、体制及管理方式等问题，甚至涉及到人的观念和人们的心理。它直接影响到项目的成败。

2.软件的分类

(1) 按软件的功能进行划分：

• 系统软件：能与计算机硬件紧密配合在一起，使计算机系统各个部件、相关的软件和数据协调、高效地工作的软件。例如，操作系统、数据库管理系统、设备驱动程序以及通信处理程序等。

• 支撑软件：是协助用户开发软件的工具性软件，其中包括帮助程序人员开发软件产品的工具，也包括帮助管理人员控制开发的进程的工具。

• 应用软件：是在特定领域内开发，为特定目的服务的一类软件。

(2) 按软件规模进行划分：

按开发软件所需的人力、时间以及完成的源程序行数，可确定六种不同规模的软件。

表 1.1 软件规模的分类

类别	参加人员数	研制期限	产品规模（源程序行数）
微型	1	1～4 周	0.5k
小型	1	1～6 月	1k～2k
中型	2～5	1～2 年	5k～50k
大型	5～20	2～3 年	50k～100k
甚大型	100～1000	4～5 年	1M(=1000k)
极大型	2000～5000	5～10 年	1M～10M

规模大、时间长、很多人参加的软件项目，其开发工作必须要有软件工程的知识做指导。而规模小、时间短、参加人员少的软件项目也得有软件工程概念，遵循一定的开发规范。其基本原则是一样的，只是对软件工程技术依赖的程度不同而已。

(3) 按软件工作方式划分：

• 实时处理软件：指在事件或数据产生时，立即予以处理，并及时反馈信号，控制需要监测和控制的过程的软件。主要包括数据采集，分析，输出三部分。

• 分时软件：允许多个联机用户同时使用计算机。

• 交互式软件：能实现人机通信的软件。

• 批处理软件：把一组输入作业或一批数据以成批处理的方式一次运行，按顺序逐个处理完的软件。

(4) 按软件服务对象的范围划分：

·项目软件：也称定制软件，是受某个特定客户（或少数客户）的委托，由一个或多个软件开发机构在合同的约束下开发出来的软件。例如军用防空指挥系统、卫星控制系统。 ·产品软件：是由软件开发机构开发出来直接提供给市场，或是为千百个用户服务的软件。例如，文字处理软件、文本处理软件、财务处理软件、人事管理软件等。

(5) 按使用的频度进行划分：

有的软件开发出来仅供一次使用。例如用于人口普查、工业普查的软件。另外有些软件具有较高的使用频度，如天气预报软件。

(6) 按软件失效的影响进行划分：

有的软件在工作中出现了故障，造成软件失效，可能给软件整个系统带来的影响不大。有的软件一旦失效。可能酿成灾难性后果。例如财务金融、交通通信、航空航天等软件。我们称这类软件为关键软件。

3.软件的发展和软件危机

自 20 世纪 40 年代中出现了世界上第一台计算机以后，就有了程序的概念。其后经历了几十年的发展，计算机软件经历了三个发展阶段：

·程序设计阶段，约为 50 至 60 年代

·程序系统阶段，约为 60 至 70 年代

·软件工程阶段，约为 70 年代以后

几十年来最根本的变化体现在：

(1) 人们改变了对软件的看法。50 年代到 60 年代时，程序设计曾经被看做是一种任人发挥创造才能的技术领域。当时人们认为，写出的程序只要能在计算机上得出正确的结果，程序的写法可以不受任何约束。随着计算机的广泛使用，人们要求这些程序容易看懂、容易使用，并且容易修改和扩充。于是，程序便从个人按自己意图创造的“艺术品”转变为能被广大用户接受的工程化产品。

(2) 软件的需求是软件发展的动力。早期的程序开发者只是为了满足自己的需要，这种自给自足的生产方式仍然是其低级阶段的表现。进入软件工程阶段以后，软件开发的成果具有社会属性，它要在市场中流通以满足广大用户的需要。

(3) 软件工作的范围从只考虑程序的编写扩展到涉及整个软件生存周期。

在软件技术发展的第二阶段，随着计算机硬件技术的进步，要求软件能与之相适应。然而软件技术的进步一直未能满足形势发展提出的要求。致使问题积累起来，形成了日益尖锐的矛盾。这就导致了软件危机。问题归结起来有：

(1) 缺乏软件开发的经验和有关软件开发数据的积累,使得开发工作的计划很难制定。致使经费预算常常突破，进度计划无法遵循，开发完成的期限一拖再拖。

(2) 软件需求，在开发的初期阶段提得不够明确，或是未能得到确切的表达。开发工作开始后，软件人员和用户又未能及时交换意见，造成开发后期矛盾的集中暴露。

(3) 开发过程没有统一的、公认的方法论和规范指导，参加的人员各行其事。加之设计和实现过程的资料很不完整；或忽视了每个人工作与其他人的接口，使得软件很难维护。

(4) 未能在测试阶段充分做好检测工作，提交用户的软件质量差，在运行中暴露出大量的问题。

如果这些障碍不能突破，进而摆脱困境，软件的发展是没有出路的。

4. 软件工程过程和软件生存周期

许多计算机和软件科学家尝试,把其它工程领域中行之有效的工程学知识运用到软件开发工作中来。经过不断实践和总结，最后得出一个结论：按工程化的原则和方法组织软件开发工作是有效的，是摆脱软件危机的一个主要出路。

(1) 软件工程过程 (Software Engineering Process)

软件工程过程是为获得软件产品,在软件工具支持下由软件工程师完成的一系列软件工程活动。软件工程过程通常包含四种基本的过程活动：

• P (Plan) : 软件规格说明。规定软件的功能及其运行的限制;

• D (Do) : 软件开发。产生满足规格说明的软件;

• C (Check) : 软件确认。确认软件能够完成客户提出的要求;

• A (Action) : 软件演进。为满足客户的变更要求，软件必须在使用的过程中演进。

事实上,软件工程过程是一个软件开发机构针对某一类软件产品为自己规定的工作步骤，它应当是科学的、合理的，否则必将影响到软件产品的质量。

(2) 软件生存周期 (life cycle)

正如同任何事物一样，软件也有一个孕育、诞生、成长、成熟、衰亡的生存过程。我们称其为计算机软件的生存周期。根据这一思想，把上述基本的过程活动进一步展开，可以得到软件生存周期的六个步骤。

• 制定计划 ： 确定要开发软件系统的总目标，给出它的功能、性能、可靠性以及接口等方面的要求；研究完成该项软件任务的可行性，探讨解决问题的可能方案；制定完成开发任务的实施计划，连同可行性研究报告，提交管理部门审查。

• 需求分析 ： 对待开发软件提出的需求进行分析并给出详细的定义。编写出软件需求说明书及初步的用户手册，提交管理机构评审。

• 软件设计 ： 把已确定了的各项需求转换成一个相应的体系结构。进而对每个模块要完成的工作进行具体的描述。编写设计说明书，提交评审。

• 程序编写 ： 把软件设计转换成计算机可以接受的程序代码。

• 软件测试 ： 在设计测试用例的基础上检验软件的各个组成部分。

• 运行 / 维护 ： 已交付的软件投入正式使用，并在运行过程中进行适当的维护。

(3) 软件生存周期模型

软件生存周期模型是从软件项目需求定义直至软件经使用后废弃为止，跨越整个生存周期的系统开发、运作和维护所实施的全部过程、活动和任务的结构框架。

a) 瀑布模型 ： 瀑布模型规定了各项软件工程活动，包括：制定开发计划，进行需求分析和说明，软件设计，程序编码。测试及运行维护，参看图 1.2。并且规定了它们自上而下，相互衔接的固定次序，如同瀑布流水，逐级下落。

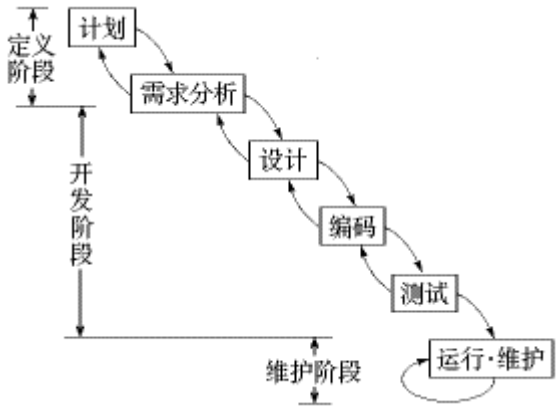


图 1.2 软件生存周期的瀑布模型

然而软件开发的实践表明，上述各项活动之间并非完全是自上而下，呈线性图式。实际情况是，每项开发活动均处于一个质量环（输入-处理-输出-评审）中。只有当其工作得到确认，才能继续进行下一项活动，在图 1.2 中用向下的箭头表示；否则返工，在图 1.2 中由向上的箭头表示。

b）演化模型 ：由于在项目开发的初始阶段人们对软件的需求认识常常不够清晰，因而使得开发项目难于做到一次开发成功，出现返工再开发在所难免。因此，可以先做试验开发，其目标只是在于探索可行性，弄清软件需求；然后在此基础上获得较为满意的软件产品。通常把第一次得到的试验性产品称为“原型”。

c）螺旋模型 ：对于复杂的大型软件，开发一个原型往往达不到要求。螺旋模型将瀑布模型与演化模型结合起来，并且加入两种模型均忽略了的风险分析。螺旋模型沿着螺线旋转，如图 1.3 所示，在笛卡尔坐标的四个象限上分别表达了四个方面的活动，即：

- 制定计划——确定软件目标，选定实施方案，弄清项目开发的限制条件；
- 风险分析——分析所选方案，考虑如何识别和消除风险；
- 实施工程——实施软件开发
- 客户评估——评价开发工作，提出修正建议。

沿螺线自内向外每旋转一圈便开发出更为完善的一个新的软件版本。

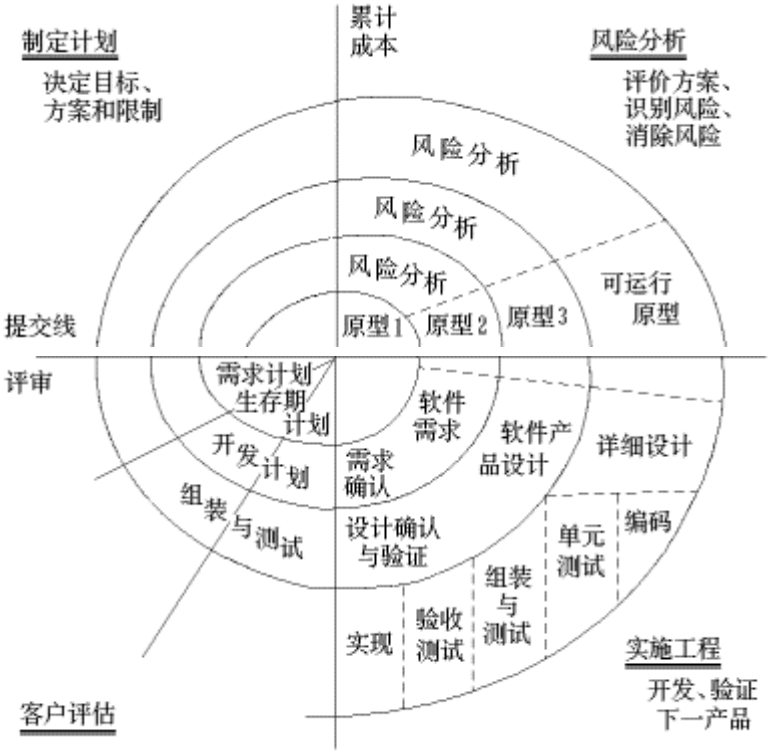


图 1.3 螺旋模型

d）喷泉模型 ：喷泉模型对软件复用和生存周期中多项开发活动的集成提供了支持，主要支持面向对象的开发方法。“喷泉”一词本身体现了迭代和无间隙特性。系统某个部分常常重复工作多次，相关功能在每次迭代中随之加入演进的系

统。所谓无间隙是指在开发活动，即分析、设计和编码之间不存在明显的边界。如图 1.4 所示。

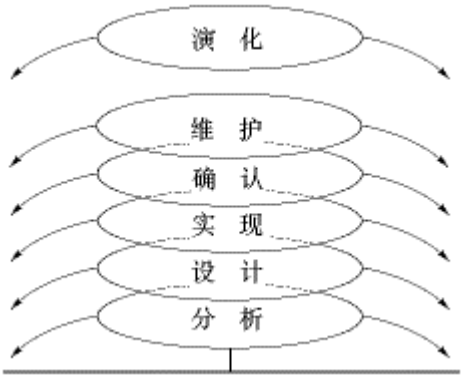


图 1.4 喷泉模型

e) 智能模型 ：智能模型是基于知识的软件开发模型，它综合了上述若干模型，并把专家系统结合在一起。该模型应用基于规则的系统，采用归约和推理机制，帮助软件人员完成开发工作，并使维护在系统规格说明一级进行。

5. 软件工程的目标和原则

(1) 软件工程的定义

Fritz Bauer 曾经为软件工程下了定义：“软件工程是为了经济地获得能够在实际机器上有效运行的可靠软件而建立和使用的一系列完善的工程化原则。”

1983 年 IEEE 给出的定义为：“软件工程是开发、运行、维护和修复软件的系统方法”，其中，“软件”的定义为：计算机程序、方法、规则、相关的文档资料以及在计算机上运行时所必需的数据。

后来尽管又有一些人提出了许多更为完善的定义，但主要思想都是强调在软件开发过程中需要应用工程化原则的重要性。

软件工程包括三个要素：方法、工具和过程。

软件工程方法为软件开发提供了“如何做”的技术。它包括了多方面的任务，如项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法过程的设计、编码、测试以及维护等。

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，这些软件工具集成起来，建立起称之为计算机辅助软件工程(CASE)的软件开发支撑系统。CASE 将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起来形成一个软件工程环境。

软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理、及软件开发各个阶段完成的里程碑。

(2) 软件工程项目的基本目标

组织实施软件工程项目，最终希望得到项目的成功。所谓成功指的是达到以下几个主要的目标：

• 付出较低的开发成本；

• 达到要求的软件功能；

• 取得较好的软件性能；

• 开发的软件易于移植；

• 需要较低的维护费用；

• 能按时完成开发工作，及时交付使用。

在具体项目的实际开发中，企图让以上几个目标都达到理想的程度往往是非常困难的。

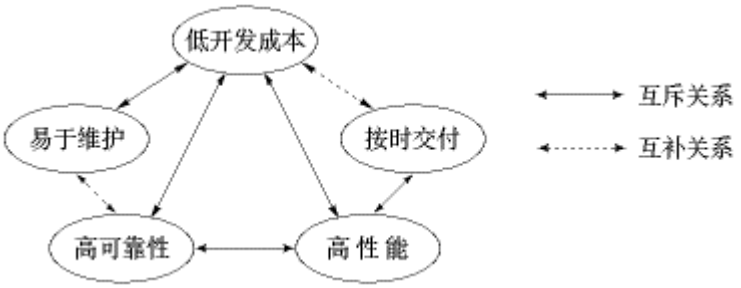


图 1.5 软件工程目标之间的关系

图 1.5 表明了软件工程目标之间存在的相互关系。其中有些目标之间是互补关系，例如，易于维护和高可靠性之间，低开发成本与按时交付之间。还有一些目标是彼此互斥的，例如，低开发成本与软件可靠性之间，提高软件性能与软件可移植性之间，就存在冲突。

(3) 软件工程的原则

以上的软件工程基本目标适合于所有的软件工程项目。为达到这些目标，在软件开发过程中必须遵循下列软件工程原则。

• 抽象 ： 抽取事物最基本的特性和行为，忽略非基本的细节。采用分层次抽象，自顶向下、逐层细化的办法控制软件开发过程的复杂性。

·信息隐蔽 ：将模块设计成“黑箱”，实现的细节隐藏在模块内部，不让模块的使用者直接访问。这就是信息封装，使用与实现分离的原则。使用者只能通过模块接口访问模块中封装的数据。

·模块化 ：模块是程序中逻辑上相对独立的成分，是独立的编程单位，应有良好的接口定义。如 C 语言程序中的函数过程，C++语言程序中的类。模块化有助于信息隐蔽和抽象，有助于表示复杂的系统。

·局部化 ：要求在一个物理模块内集中逻辑上相互关联的计算机资源，保证模块之间具有松散的耦合，模块内部具有较强的内聚。这有助于控制解的复杂性。

·确定性 ：软件开发过程中所有概念的表达应是确定的、无歧义性的、规范的。这有助于人们之间在交流时不会产生误解、遗漏，保证整个开发工作协调一致。 ·一致性 ：整个软件系统（包括程序、文档和数据）的各个模块应使用一致的概念、符号和术语。程序内部接口应保持一致。软件和硬件、操作系统的接口应保持一致。系统规格说明与系统行为应保持一致。用于形式化规格说明的公理系统应保持一致。

·完备性 ：软件系统不丢失任何重要成分，可以完全实现系统所要求功能的程度。为了保证系统的完备性，在软件开发和运行过程中需要严格的技术评审。

·可验证性 ：开发大型的软件系统需要对系统自顶向下、逐层分解。系统分解应遵循系统易于检查、测试、评审的原则，以确保系统的正确性。

使用一致性、完备性和可验证性的原则可以帮助人们实现一个正确的系统。

三、例题分析

【例 1】软件是计算机系统中与硬件相互依存的另一部分，它是包括（A）、（B）及（C）的完整集合。其中，（A）是按事先设计的功能和性能要求执行的指令序列。（B）是使程序能够正确操纵信息的数据结构。（C）是与程序开发、维护和使用有关的图文材料。

供选择的答案：

A~C. ① 软件 ② 程序 ③ 代码 ④ 硬件

⑤ 文档 ⑥ 外设 ⑦ 数据 ⑧ 图表 答案：A. ②, B. ⑦, C. ⑤

分析：软件是计算机系统中与硬件相互依存的另一部分，它是包括程序、数据及相关文档的完整集合。其中，程序是按事先设计的功能和性能要求执行的指令序列。数据是使程序能够正确操纵信息的数据结构。文档是与程序开发、维护和使用有关的图文材料。需要注意的是，程序与算法在含义上有不同：算法的每一条指令必须是最基本的、必须通过有限步做完，而程序没有这个要求。

【例 2】开发软件时对提高软件开发人员工作效率至关重要的是(A)。软件工程中描述生存周期的瀑布模型一般包括计划、(B)、设计、编码、测试、维护等几个阶段，其中设计阶段在管理上又可以依次分成(C)和(D)两步。

供选择的答案：

A. ① 程序开发环境 ② 操作系统的资源管理功能

③ 程序人员数量 ④ 计算机的并行处理能力

B. ① 需求分析 ② 需求调查 ③ 可行性分析 ④ 问题定义

C、D. ① 方案设计 ② 代码设计 ③ 概要设计 ④ 数据设计

⑤ 运行设计 ⑥ 详细设计 ⑦ 故障处理设计 ⑧ 软件体系结构设计

答案：A. ①, B. ①, C. ③, D. ⑥ 分析：有合适的程序开发环境可以提供有用的工具，大大提高开发人员的工作效率。

软件工程中描述软件生存周期的瀑布模型一般包括计划、需求分析、设计、编码、测试和运行维护等六个阶段。需求分析完成对软件产品在功能、性能、用户接口、运行环境、可靠性、安全性、开发资源、开发进度、开发成本等方面的需求定义。问题定义、可行性分析、需求调查都可能是需求分析中要做的一部分工作。

软件设计在管理上划分为概要设计与详细设计两个步骤。概要设计的目标是建立软件的体系结构，完成全局数据结构设计，同时进行处理方式设计、运行配置设计、出错处理设计、故障恢复设计等。详细设计是对每一个模块的操作的控制流程和局部数据结构进行设计。

【例 3】从供选择的答案中选出适当字句填入下列关于软件发展过程的叙述中的( )内。

有人将软件的发展过程划分为 4 个阶段：

第一阶段(1950～1950 年代末)称为“程序设计的原始时期”，这时既没有(A)，也没有(B)，程序员只能用机器指令编写程序。

第二阶段（1950 年代末～1960 年代末）称为“基本软件期”。出现了(A)，并逐渐普及。随着(B)的发展，编译技术也有较大的发展。

第三阶段（1960 年代末～1970 年代中期）称为“程序设计方法时代”。这一时期，与硬件费用下降相反，软件开发费急剧上升。人们提出了(C)和(D)等程序设计方法，设法降低软件的开发费用。

第四阶段（1970 年代中期～现在）称为“软件工程时期”。软件开发技术不再仅仅是程序设计技术，而是包括了与软件开发的各个阶段，如（ E ）、（ F ）、编码、单元测试、综合测试、（ G ）及其整体有关的各种管理技术。

供选择的答案：

A ~ D：① 汇编语言 ② 操作系统 ③ 虚拟存储器概念 ④ 高级语言

⑤ 结构式程序设计 ⑥ 数据库概念 ⑦ 固件 ⑧ 模块化程序设计

E ~ G：① 使用和维护 ② 兼容性的确认 ③ 完整性的确认 ④ 设计

⑤ 需求定义 ⑥ 图象处理

答案：A. ①, B. ④, C. ⑤, D. ⑧, E. ⑤, F. ④, G. ①。注意，C 与 D 的答案顺序可互换。

分析：在软件发展的早期，汇编语言和高级语言尚未出现，人们只能用机器指令来编写程序。为了好读，有的指令系统可以用八进制代码书写。其后，由于汇编语言的出现，人们摆脱了繁重的地址分配等工作，可以用符号编程。随着高级语言的出现和普及，人们用近似于自然语言的语句编写程序，大大减轻了程序员的负担。高级语言的处理，从解释执行到编译执行，处理效率和存储利用率不断提高。 结构化程序设计和模块化程序设计是科学家为解决软件危机，借用其它领域的技术改进程序设计方法而提出来的。由于这些技术的使用，提高了程序的可读性、局部性、抽象性、清晰性、简单性、确定性、一致性等，降低了程序开发的费用。后来发展到软件工程阶段，明确地划分了软件开发阶段，规范了软件开发过程，明确了各个阶段的任务以及应交付的成果和里程碑，使得软件开发逐步达到工程化和标准化。

四、习题

【1-1】软件工程过程有哪几个基本过程活动？试说明之。

【1-2】试说明“软件生存周期”的概念。

【1-3】试论述瀑布模型软件开发方法的基本过程。

【1-4】软件工程是开发、运行、维护和修复软件的系统化方法，它包含哪些要素？试说明之。

【1-5】软件工程学的基本原则有哪些？试说明之。

【1-6】有人说：软件开发时，一个错误发现得越晚，为改正它所付出的代价就越大。对否？请解释你的回答。

五、习题解答

【1-1】软件工程过程的基本过程活动有 4 步：

• P (Plan) : 软件规格说明。规定软件的功能及其运行的限制；

• D (Do) : 软件开发。产生满足规格说明的软件；

• C (Check) : 软件确认。确认软件能够完成客户提出的要求；

• A (Action) : 软件演进。为满足客户的变更要求，软件必须在使用的过程中演进。

【1-2】软件与任何一个事物一样，有它的孕育、诞生、成长、成熟、衰亡的生存过程。这就是软件的生存周期。它主要分为 6 个阶段：软件项目计划、软件需求分析和定义、软件设计、程序编码、软件测试，以及运行维护。

(1) 软件项目计划 ：在这一步要确定软件工作范围，进行软件风险分析，预计软件开发所需要的资源，建立成本与进度的估算。根据有关成本与进度的限制分析项目的可行性。

(2) 软件需求分析和定义 ：在这一步详细定义分配给软件的系统元素。可以用以下两种方式中的一种对需求进行分析和定义。一种是正式的信息域分析，可用于建立信息流和信息结构的模型，然后逐渐扩充这些模型成为软件的规格说明。另一种是软件原型化方法，即建立软件原型，并由用户进行评价，从而确定软件需求。

(3) 软件设计 ：软件的设计过程分两步走。第一步进行概要设计，以结构设计和数据设计开始，建立程序的模块结构，定义接口并建立数据结构。此外，要使用一些设计准则来判断软件的质量。第二步做详细设计，考虑设计每一个模块部件的过程描述。经过评审后，把每一个加细的过程性描述加到设计规格说明中去。

(4) 程序编码 ：在设计完成之后，用一种适当的程序设计语言或 CASE 工具生成源程序。应当就风格及清晰性对代码进行评审，而且反过来应能直接追溯到详细设计描述。

(5) 软件测试 ：单元测试检查每一单独的模块部件的功能和性能。组装测试提供了构造软件模块结构的手段，同时测试其功能和接口。确认测试检查所有的需求是否都得到满足。在每一个测试步骤之后，要进行调试，以诊断和纠正软件的故障。

(6) 软件维护 ：为改正错误，适应环境变化及功能增强而进行的一系列修改活动。与软件维护相关联的那些任务依赖于所要实施的维护的类型。

【1-3】瀑布模型规定了各项软件工程活动，包括：制定软件项目计划，进行需求分析和定义，软件设计，程序编码，测试及运行维护。并且规定了它们自上而下，相互衔接的固定次序，如同瀑布流水，逐级下落。然而软件开发的实践表明，上述各项活动之间并非完全是自上而下，呈线性图式。实际情况是，每项开发活动均应具有以下特征：

(1) 从上一项活动接受本项活动的工作对象，做为输入；

(2) 利用这一输入实施本项活动应完成的内容；

(3) 给出本项活动的工作成果，做为输出传给下一项活动；

(4) 对本项活动实施的工作进行评审。若其工作得到确认，则继续进行下一项活动，否则返回前项，甚至更前项的活动进行返工。

【1-4】软件工程包括三个要素：方法、工具和过程。

软件工程方法为软件开发提供了“如何做”的技术。它包括了多方面的任务，如项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法过程的设计、编码、测试以及维护等。软件工程方法常采用某一种特殊的语言或图形的表达方法及一套质量保证标准。

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，已经能够支持上述的软件工程方法。特别地，已经有人把诸多的软件工具集成起来，使得一种工具产生的信息可以为其它的工具所使用，这样建立起一种被称之为计算机辅助软件工程(CASE)的软件开发支撑系统。CASE将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起来形成一个软件工程环境。

软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理、及软件开发各个阶段完成的里程碑。

【1-5】在软件开发过程中必须遵循下列软件工程原则。

• 抽象 ： 采用分层次抽象，自顶向下、逐层细化的办法进行功能分解和过程分解，可以由抽象到具体、由复杂到简单，逐步得到问题的解。

• 信息隐蔽 ： 遵循信息封装，使用与实现分离的原则，将模块设计成“黑箱”，可以将实现的细节隐藏在模块内部，使用者只能通过模块接口访问模块中封装的数据。

• 模块化 ： 按模块划分系统的体系结构，使得各模块间有良好的接口。这样有助于信息隐蔽和抽象，有助于表示复杂的系统。

• 局部化 ： 按抽象数据类型思想及问题域中的概念来建立模块，确保模块之间低耦合，模块内部高内聚。这有助于控制解的复杂性。

• 确定性 ： 软件开发过程中所有概念的表达应是确定的、无歧义性的、规范的。这有助于人们之间的沟通，保证整个开发工作协调一致。

• 一致性 ： 强调软件开发过程的标准化、统一化。包括文档格式的一致，工作流程的一致，内、外部接口的一致，系统规格说明与系统行为的一致等。

• 完备性 ： 软件系统不丢失任何重要成分，可以完全实现系统所要求功能。

• 可验证性 ： 开发大型的软件系统需要对系统自顶向下、逐层分解。系统分解应遵循系统易于检查、测试、评审的原则，以确保系统的正确性。

【1-6】软件开发时，一个错误发现得越晚，为改正它所付出的代价就越大。这个说法是对的。在 1970 年代，GTE、TRW 和 IBM 等三家公司对此问题做了独立研究，最后它们得到相似的结论：

阶段	需求分析	软件设计	程序编码	单元测试	验收测试	维 护
相对修复代价	0.1~0.2	0.5	1	2	5	20

从表中可以看出，在需求分析阶段检查和修复一个错误所需的代价只有编码阶段所需代价的 1/5 到 1/10，而在维护阶段做同样的工作所付出的代价却是编码阶段的 20 倍。

第二章 软件需求分析

一、复习要求

1. 了解软件需求的目标和任务。

2. 了解软件软件需求的获取方法。

3. 了解可行性研究的方法和可行性研究报告的主要内容。

4. 掌握结构化分析方法。

5. 了解支持需求分析的原型化方法。

6. 了解需求规格说明和需求评审的主要内容。

二、内容提要

1. 软件需求分析的目标和任务

软件需求分析的目标是深入描述软件的功能和性能，确定软件设计的约束和软件同其它系统元素的接口细节，定义软件的其它有效性需求。

需求分析阶段研究的对象是软件项目的用户要求。一方面，必须全面理解用户的各项要求，但又不能全盘接受所有的要求，另一方面，要准确地表达被接受的用户要求。只有经过确切描述的软件需求才能成为软件设计的基础。

通常软件开发项目是要实现目标系统的物理模型。作为目标系统的参考，需求分析的任务就是借助于当前系统的逻辑模型导出目标系统的逻辑模型，解决目标系统的“做什么”的问题。其实现步骤如图 2.1 所示。

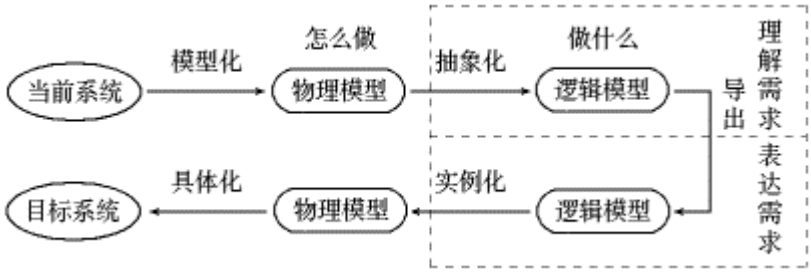


图 2.1 参考当前系统建立目标系统模型

2. 需求分析的过程

需求分析阶段的工作，可以分成以下四个方面：

(1) 问题识别

首先系统分析人员要确定对目标系统的综合要求，即软件的需求。并提出这些需求实现条件，以及需求应达到的标准。这些需求包括功能需求、性能需求、环境需求、可靠性需求、安全保密要求、用户界面需求、资源使用需求、软件成本消耗与开发进度需求，并预先估计以后系统可能达到的目标。此外，还需要注意其它非功能性的需求。如针对采用某种开发模式，确定质量控制标准、里程碑和评审、验收标准、各种质量要求的优先级等，以及可维护性方面的需求。

此外，要建立分析所需要的通信途径，以保证能顺利地对问题进行分析。分析所需的通信途径如图 2.2 所示。

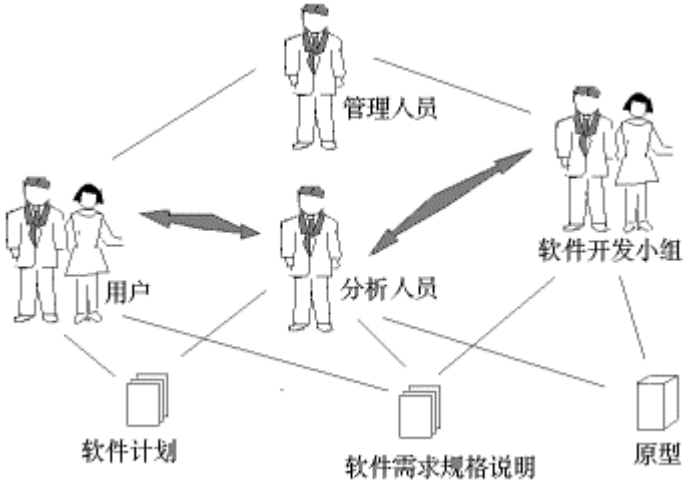


图 2.2 软件需求分析的通信途径

(2) 分析与综合

问题分析和方案的综合是需求分析的第二方面的工作。分析员必须从信息流和信息结构出发，逐步细化所有的软件功能，找出系统各元素之间的联系、接口特性和设计上的限制，判断是否存在因片面性或短期行为而导致的不合理的用户要求，是否有用户尚未提出的真正有价值的潜在要求。剔除其不合理的部分，增加其需要部分。最终综合成系统的解决方案，给出目标系统的详细逻辑模型。

(3) 编制需求分析阶段的文档

已经确定下来的需求应当得到清晰准确的描述。通常我们把描述需求的文档叫做软件需求说明书。同时，为了确切表达用户对软件的输入输出要求，还需要制定数据要求说明书及编写初步的用户手册。

(4) 需求分析评审

作为需求分析阶段工作的复查手段，应该对功能的正确性、文档的一致性、完备性、准确性和清晰性，以及其它需求给予评价。为保证软件需求定义的质量，评审应以专门指定的人员负责，并按规程严格进行。评审结束应有评审负责人的结论意见及签字。除分析员之外，用户 / 需求者，开发部门的管理者，软件设计、实现、测试的人员都应当参加评审工作。

3. 需求获取技术

需求获取技术包括两方面的工作：

§ 建立获取用户需求的方法的框架；

§ 支持和监控需求获取的过程的机制。

获取用户需求的主要方法是调查研究。

(1) 了解系统的需求。软件开发常常是系统开发的一部分。仔细分析研究系统的需求规格说明，对软件的需求获取是很有必要的。

(2) 市场调查。了解市场对待开发软件有什么样的要求；了解市场上有无与待开发软件类似的系统。如果有，在功能上、性能上、价格上情况如何。

(3) 访问用户和用户领域的专家。把从用户那里得到的信息作为重要的原始资料进行分析；访问用户领域的专家所得到的信息将有助于对用户需求的理解。

(4) 考察现场。了解用户实际的操作环境、操作过程和操作要求。对照用户提交的问题陈述，对用户需求可以有更全面、更细致的认识。 在做调查研究时，可以采取如下的调查方式：

§ 制定调查提纲，向不同层次的用户发调查表。

§ 按用户的不同层次，分别召开调查会，了解用户对待开发系统的想法和建议。

§ 向用户领域的专家或在关键岗位上工作的人个别咨询。

§ 实地考察，跟踪现场业务流程。

§ 查阅与待开发系统有关的资料。

§ 使用各种调查工具，如数据流图、任务分解图、网络图等。

为了能够有效地获取和理清用户需求，应当打破用户（需方）和开发者（供方）的界限，共同组成一个联合小组，发挥各自的长处，协同工作。

4. 可行性研究和可行性研究报告

(1) 可行性研究

这是在软件项目计划阶段应该做的事情，包括四个方面的研究：

§ 经济可行性 ：进行成本／效益分析。从经济角度判断系统开发是否“合算”。

§ 技术可行性 ：进行技术风险评价。从开发者的技术实力、以往工作基础、问题的复杂性等出发，判断系统开发在时间、费用等限制条件下成功的可能性。

§ 法律可行性 ：确定系统开发可能导致的任何侵权、妨碍和责任。

§ 方案的选择 ：评价系统或产品开发的几个可能的候选方案。最后给出结论意见。

(2) 经济可行性

分析员需要进行成本/效益分析。所谓成本，包括：① 购置并安装软、硬件及有关设备的费用；② 系统开发费用；③ 系统安装、运行及维护的费用；④ 人员培训费用。而效益是指：① 系统为用户增加的收入或为用户节省的开支，这是有形的效益；② 给潜在用户心理上造成的影响，这是无形的效益。它可以转化为有形的效益。

(3) 技术可行性

分析员需要根据系统的功能、性能需求，建立系统模型。然后对此模型进行一系列的试验、评审和修改。最后由项目管理人员作出是否进行系统开发的决定。

如果开发技术风险很大，或者模型演示表明当前采用的技术和方法不能实现系统预期的功能和性能，或者系统的实现不支持各子系统的集成，则项目管理人员可以作出停止系统开发的决定。

(4) 方案的选择

分析员考虑问题解决的方案。一般采用将一个大而复杂的系统分解为若干个子系统的办法来降低解的复杂性。如何进行系统分解、如何定义各子系统的功能、性能和界面，实现方案不唯一。可以采用折衷的方法，反复比较各个方案的成本/效益，选择可行的方案。

(5) 可行性研究报告

可行性报告的形式可以有多种，但最重要的内容应当有：

Ⅰ. 项目背景：① 问题描述 ② 实现环境 ③ 限制条件

Ⅱ. 管理概要和建议：① 重要的研究结果 ② 说明 ③ 建议 ④ 影响
Ⅲ. 候选方案：① 候选系统的配置 ② 最终方案的选择标准
Ⅳ. 系统描述：① 系统工作范围的简要说明 ② 被分配系统元素的可行性
Ⅴ. 经济可行性（成本－效益分析）：① 经费概算 ② 预期的经济效益

Ⅵ. 技术可行性（技术风险评价）：① 技术实力 ② 已有工作基础 ③ 设备条件
Ⅶ. 法律可行性：① 系统开发可能导致的侵权，违法和责任
Ⅷ. 用户使用可行性：① 用户单位的行政管理，工作制度 ② 使用人员的素质
Ⅸ. 其它与项目有关的问题：① 其它方案介绍 ② 未来可能的变化

可行性研究报告首先由项目负责人审查（审查内容是否可靠），再上报给上级主管审阅（估价项目的地位）。从可行性研究应当得出“行或不行”的决断。当然，在以后的开发阶段，还要其它“行还是不行”的决定。

5. 结构化分析方法

结构化分析方法最初由 Douglas Ross 提出，由 DeMarco 推广，由 Ward 和 Mellor 以及后来的 Hatley 和 Pirbhai 扩充，形成了今天的结构化分析方法的框架。

结构化分析方法是一种建模技术。它建立的分析模型如图 2.3 所示。

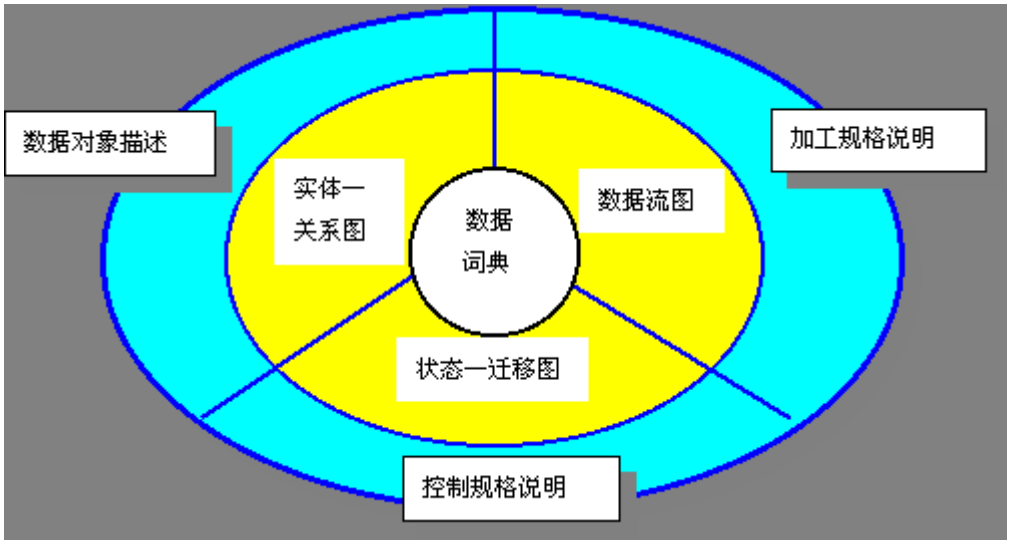


图 2.3 分析模型的结构

在模型的核心是数据词典，它描述了所有的在目标系统中使用的和生成的数据对象。围绕着这个核心的有三种图：实体—关系图(ERD)描述数据对象及数据对象之间的关系；数据流图(DFD)描述数据在系统中如何被传送或变换，以及描述如何对数据流进行变换的功能（子功能）；状态—迁移图(STD)描述系统对外部事件如何响应，如何动作。

因此，ERD 用于数据建模，DFD 用于功能建模，STD 用于行为建模。

(1) 数据建模

数据模型包括三种互相关联的信息：数据对象，描述对象的属性，描述对象间相互连接的关系。

§ 数据对象 ：是需被目标系统所理解的复合信息的表示。所谓复合信息是具有若干不同特征或属性的信息。

数据对象可以是外部实体（如显示器），事物（如报表或显示），角色（如教师或学生），行为（如一个电话呼叫）或事件（如单击鼠标左键），组织单位（如研究生院），地点（如注册室）或结构（如文件）。

数据对象只封装了数据，没有包含作用于这些数据上的操作。这与面向对象范型中的类和对象不同。具有相同特征的数据对象组成的集合仍然称为数据对象，其中的某一个对象叫做该数据对象的一个实例。

§ 属性 ：定义了数据对象的特征。它可用来：① 为数据对象的实例命名；② 描述这个实例；③ 建立对另一个数据对象的另一个实例的引用。如学生数据对象的属性可以有学号、姓名、性别、出生年月、籍贯等。为了唯一地标识数据对象的某一个实例，定义数据对象中的一个属性或几个属性为关键码(key)，书写为_id,例如在“学生”数据对象中用“学号”做关键码,它可唯一地标识一个“学生”数据对象中的实例。

§ 关系 ：各个数据对象的实例之间有关联。如一个学生“张鹏”选修两门课程“软件工程”与“计算机网络”，学生与课程的实例通过“选修”关联起来。实例的关联有三种：① 一对一(1:1)；② 一对多(1:m)；③ 多对多(n:m)。这种实例的关联称为“基数”。基数表明了“重复性”。如 1 位教师带学生班的 30 位同学，就是 1:m 的关系。但也有 1 位教师带 0 位同学的情形。所以实例关联有是“可选”还是“必须”之分。用“〇”表示关系是可选的，用“｜”表示关系必须出现 1 次。如图 2.4 所示。这表明了关系的“参与性”。



图 2.4 基数与参与性

§ 实体—关系图 ：数据对象及其关系可用 ERD 表示。图 2.5 给出学生选修课程的 ERD 及描述学生属性的实体对象表。

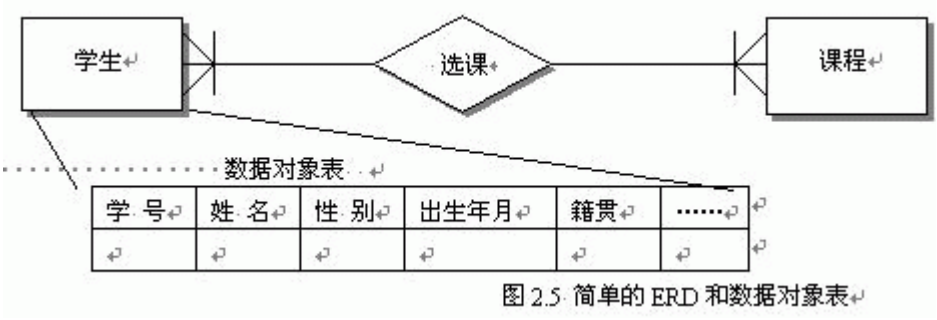


图 2.5 简单的 ERD 和数据对象表

(2) 功能建模和数据流

最初，结构化分析方法仅讨论数据流建模。目标系统被表示成如图 2.6 所示的数据变换流程图。系统的功能体现在核心的数据变换中。

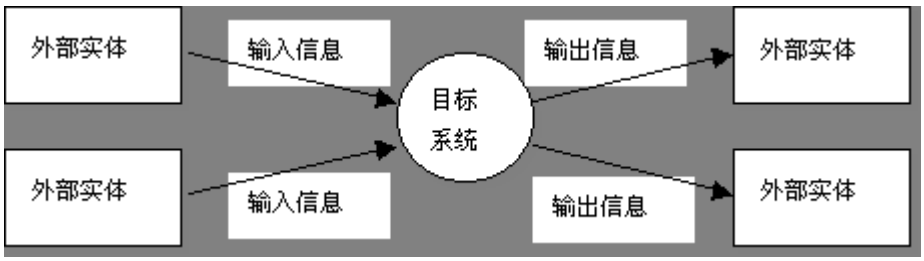


图 2.6 数据流图（DFD）

功能建模的思想就是用抽象模型的概念，按照软件内部数据传递、变换的关系，自顶向下逐层分解，直到找到满足功能要求的所有可实现的软件为止。根据 DeMarco 的论述，功能模型使用了数据流图来表达系统内数据的运动情况，而数据流的变换则用结构化英语、判定表与判定树来描述。

§ 数据流图

图 2.7 是描述储户携带存折去银行办理取款手续的数据流图。从图中可以看到，数据流图的基本图形元素有四种，如图 2.8 所示。

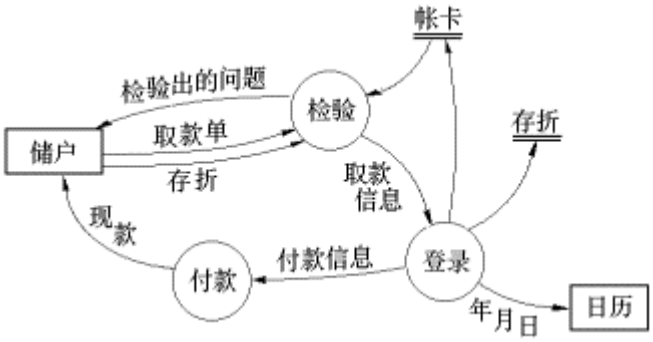


图 2.7 办理取款手续的数据流图

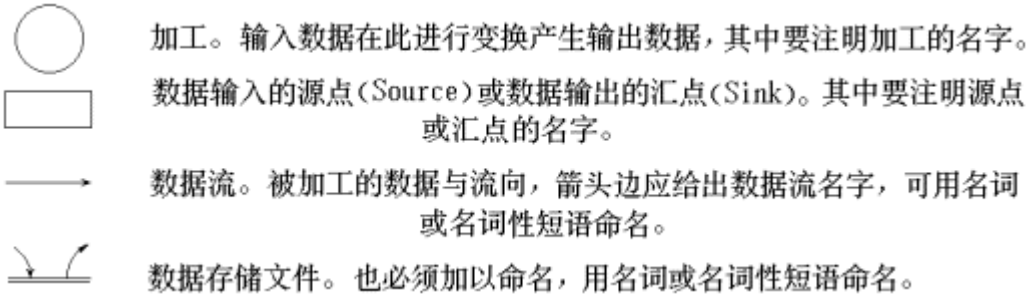


图 2.8 DFD 的基本图形符号

在数据流图中，如果有两个以上数据流指向一个加工，或是从一个加工中引出两个以上的数据流，这些数据流之间往往存在一定的关系。为表达这些关系，在这些数据流的加工可以标上不同的标记符号。所用符号及其含意在图 2.9 中给出。

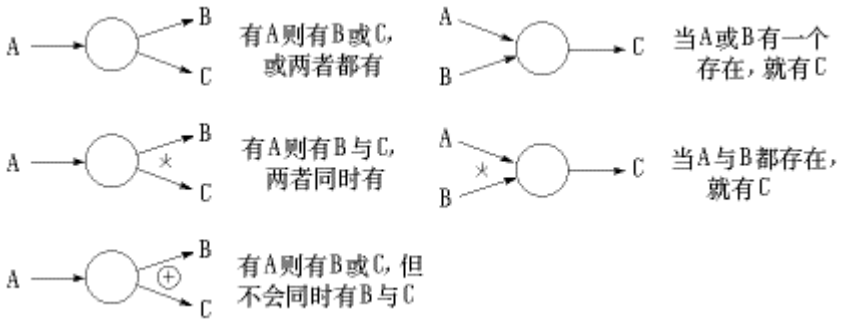


图 2.9 表明多个数据流与加工之间关系的符号

§ 分层数据流图

为了表达数据处理过程的数据加工情况，用一个数据流图是不够的。稍为复杂的实际问题，在数据流图上常常出现十几个甚至几十个加工。这样的数据流图看起来很不清楚。层次结构的数据流图能很好地解决这一问题。按照系统的层次结构进行逐步分解，并以分层的数据流图反映这种结构关系，能清楚地表达和容易理解整个系统。

图 2.10 给出分层数据流图的示例。数据处理 S 包括三个子系统 1、2、3。顶层下面的第一层数据流图为 DFD / L1。第二层数据流图 DFD / L2.1、DFD / L2.2 及 DFD / L2.3 分别是子系统 1、2 和 3 的细化。对任何一层数据流图来说，我们称它的上层图为父图，在它下一层的图则称为子图。

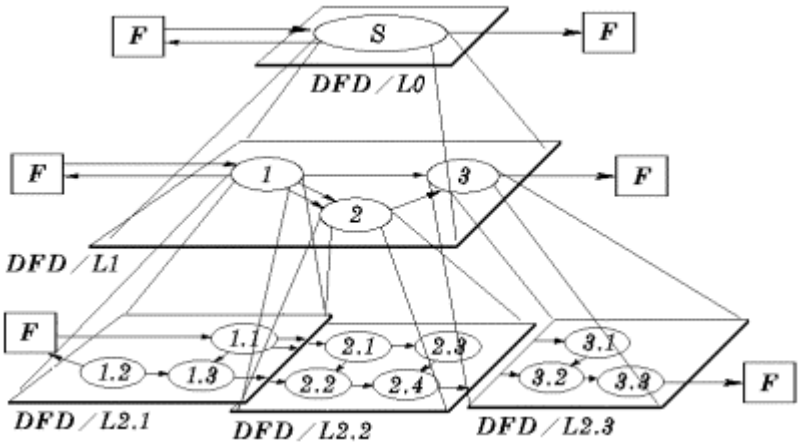


图 2.10 分层数据流图

画数据流图的基本步骤概括地说，就是自外向内，自顶向下，逐层细化，完善求精。检查和修改的原则为：

① 数据流图上所有图形符号只限于前述四种基本图形元素。

② 顶层数据流图必须包括前述四种基本元素，缺一不可。

③ 顶层数据流图上的数据流必须封闭在外部实体之间。

④ 每个加工至少有一个输入数据流和一个输出数据流。

⑤ 在数据流图中，需按层给加工框编号。编号表明该加工处在哪一层，以及上下层的父图与子图的对应关系。

⑥ 规定任何一个数据流子图必须与它上一层的一个加工对应，两者的输入数据流和输出数据流必须一致。此即父图与子图的平衡。

⑦ 可以在数据流图中加入物质流，帮助用户理解数据流图。

⑧ 图上每个元素都必须有名字。数据流和数据文件的名字应当是“名词”或“名词性短语”，表明流动的数据是什么。加工的名字应当是“名词＋宾语”，表明做什么事情。

⑨ 数据流图中不可夹带控制流。

⑩ 初画时可以忽略琐碎的细节，以集中精力于主要数据流。

§ 加工规格说明

加工规格说明用来说明 DFD 中的数据加工的加工细节。加工规格说明描述了数据加工的输入，实现加工的算法以及产生的输出。另外，加工规格说明指明了加工（功能）的约束和限制，与加工相关的性能要求，以及影响加工的实现方式的设计约束。必须注意，写加工规格说明的主要目的是要表达“做什么”，而不是“怎样做”。因此它应描述数据加工实现加工的策略而不是实现加工的细节。

目前用于写加工规格说明的工具有结构化英语、判定表和判定树。

§ 针对实时系统的 Ward & Mellor 扩展

这种扩展可以适应实时系统提出的以下要求：① 在时间连续的基础上接收或产生数据流；② 贯穿系统的控制信息和相关的控制处理；③ 在多任务的情况下可能会遇到同一个加工的多个实例；④ 系统状态以及导致系统状态迁移的机制。

图 2.11 给出的扩展的图形符号可以让分析员在描述数据流和加工的同时，描述控制流和控制加工。这些符号可以与原来的数据流图的图形符号混用。

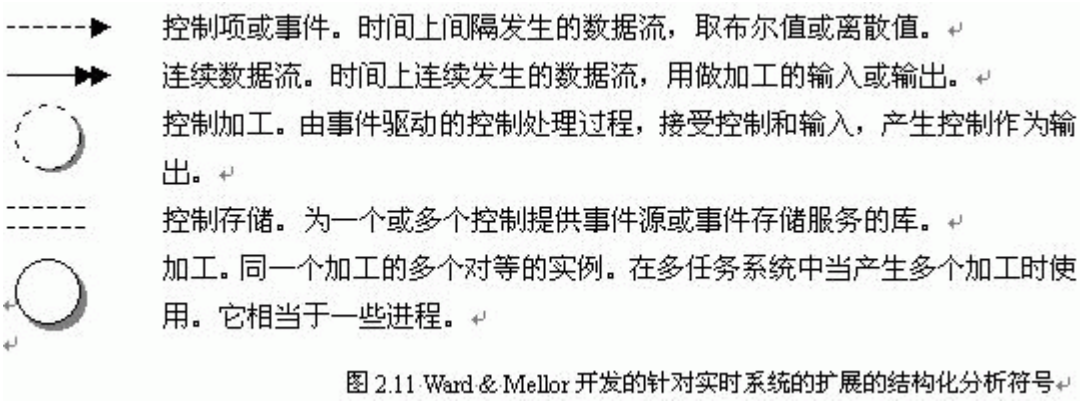


图 2.11 Ward & Mellor 开发的针对实时系统的扩展的结构化分析符号

例如，图 2.12 给出一个基于计算机的水温控制系统的处理。水温测量仪连续传送水温数据给温度监控加工模块，该加工将水温与允许波动范围进行比较，输出校正数据给温度调节装置。

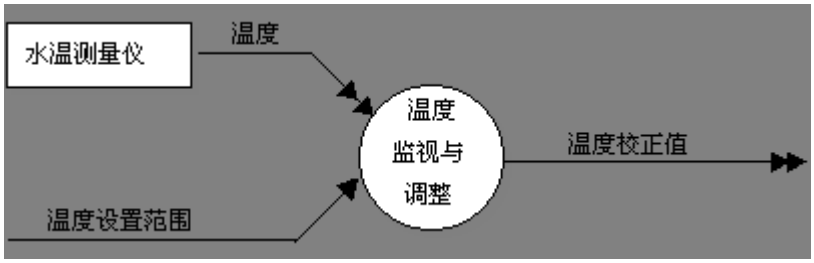


图 2.12 时间连续的数据流与普通数据流

图 2.13 给出一个制造车间的数据和控制流的顶层流图。事件可以作为普通数据加工的输入数据流，控制也可以接收普通的输入数据流。

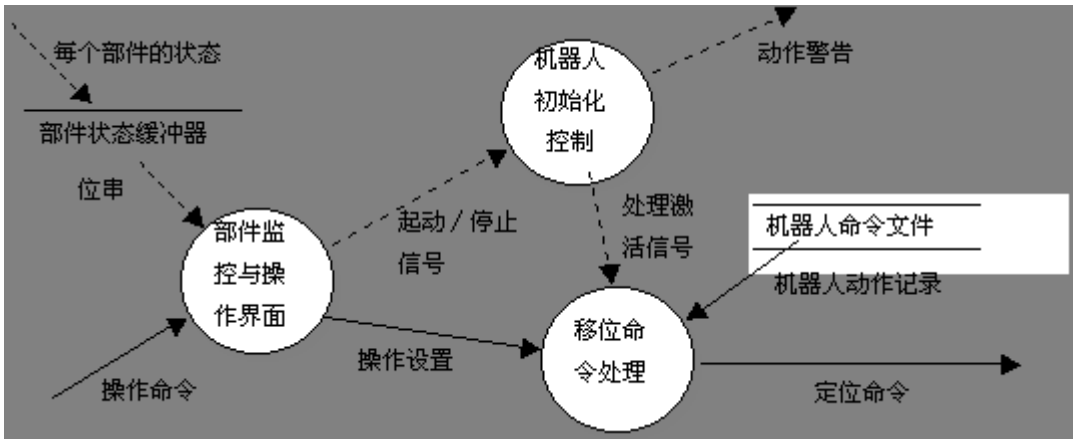


图 2.13 使用 Ward & Mellor 符号的数据流和控制流

§ Hatley 和 Pirbhai 对结构化分析技术的扩展

Ward & Pirbhai 方法主要关注面向控制的规格说明，而不是扩充数据流图的图形符号。

他们建立控制流图(CFD)以区别于数据流图(DFD)。在控制流图中的加工与数据流图中相同，但传递的是控制流而不是数据流。在控制流图中引入实短线“|”表示对控制规格说明的引用。图 2.14 给出他们建立的实时系统模型。用数据流图表示对数据和操作数据的加工；用控制流图表示事件在加工之间如何流动，说明导致各个加工激活的外部事件。

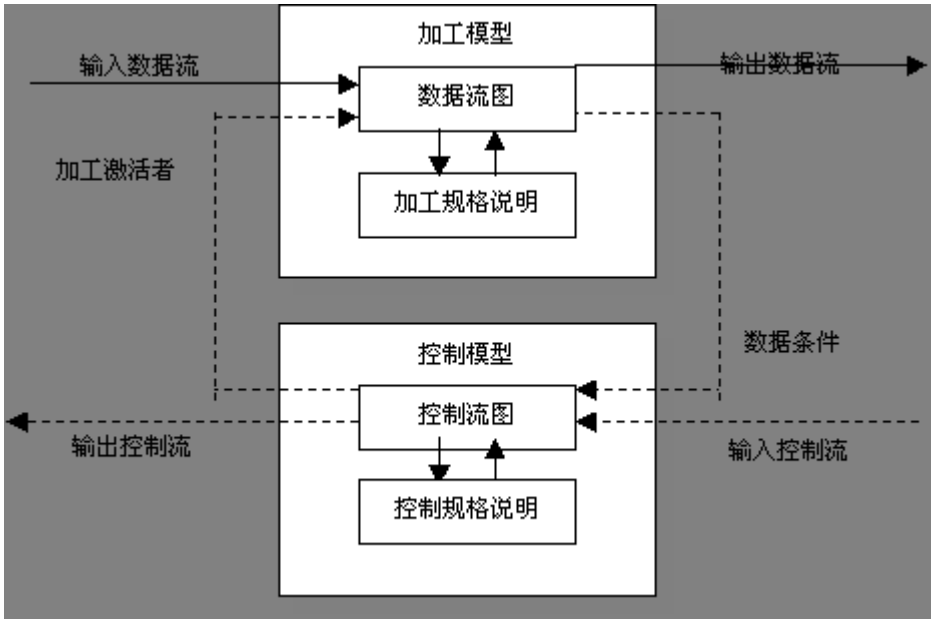


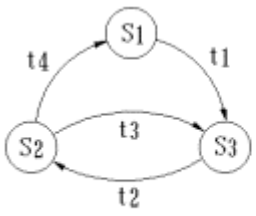
图 2.14 数据与控制之间的关系

(3) 行为建模

行为建模给出需求分析方法的所有操作原则，但只有结构化分析方法的扩充版本才提供这种建模的符号。

§ 状态—迁移图

利用如图 2.15 所示的状态—迁移图(STD)或状态—迁移表来描述系统或对象的状态，以及导致系统或对象的状态改变的事件，从而描述系统的行为。



(a) 状态迁移图

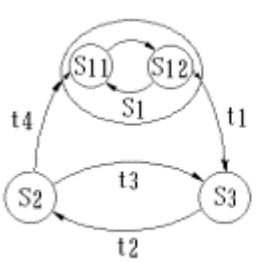
状态 / 事件	S1	S2	S3
t1	S3		
t2			S2
t3		S3	
t4		S1	

(b) 状态迁移表

图 2.15 状态—迁移图与其等价的状态—迁移表例

每一个状态代表系统或对象的一种行为模式。状态—迁移图指明系统的状态如何相应外部的信号（事件）进行推移。在状态—迁移图中，用圆圈“○”表示可得到的系统状态，用箭头“→”表示从一种状态向另一种状态的迁移。在箭头上要写上导致迁移的信号或事件的名字。 如图 2.15(a) 所示，系统中可取得的状态＝S1，S2，S3，事件＝t1，t2，t3，t4。事件 t1 将引起系统状态 S1 向状态 S3 迁移，事件 t2 将引起系统状态 S3 向状态 S2 迁移，等等。图 2.15(b) 就是与图 2.15(a) 等价的状态—迁移表。 另外，状态—迁移图指明了作为特定事件的结果（状态）。在状态中包含可能执行的行为（活动或加工）。

如果系统比较复杂，可以把状态—迁移图分层表示。例如，在确定了如图 2.16 所示那样的大状态 S1，S2，S3 之后，接下来就可把状态 S1，S2，S3 细化。在该图中对状态 S1 进行了细化。此外，在状态—迁移图中，由一个状态和一个事件所决定的下一状态可能会有多个。实际会迁移到哪一个是由更详细的内部状态和更详细的事件信息来决定的。此时，可采用状态迁移图的一种变形，如图 2.17 那样，使用加进判断框和处理框的记法。



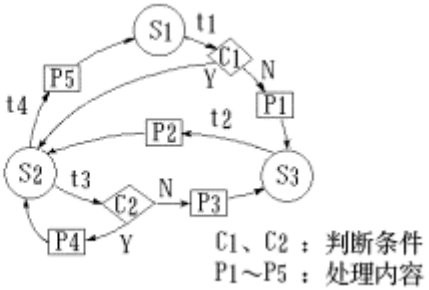


图 2.16 状态迁移图的网 图 2.17 状态迁移图的变形

§ Petri 网

Petri 网，简称 PNG (Petri Net Graph)。它适用于描述相互独立、协同操作的处理系统，即并发执行的处理系统。在软件需求分析与设计阶段都可以使用。

Petri 网是一种有向图，它有两种结点：“○”表示系统的状态。“—”或“┃”表示系统中的事件。图中的有向边表示对事件的输入，或从事件输出：“ ⇥ ”表示对事件的输入；“ ↦ ”表示事件的结果，即从事件的输出。

图 2.18 用 Petri 网描述了在一个多任务系统中的两个进程 PR1 和 PR2 使用一个公共资源 R 时，利用原语 LOCK（对资源加锁）和 UNLOCK（对资源解锁）控制 R 的使用，保证进程间的同步的例子。

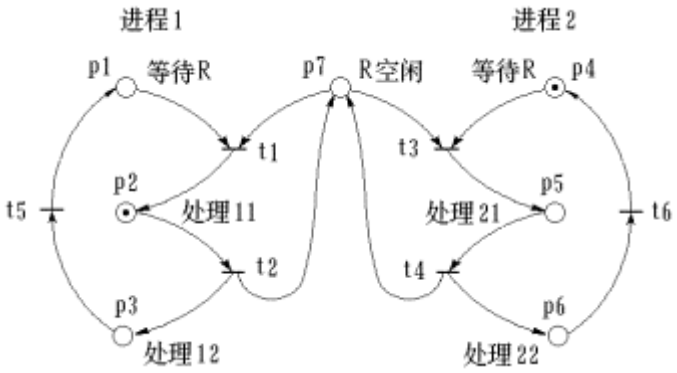


图 2.18 进程同步机制的 PNG

图中每个进程是一个数据对象，它有三个状态：等待资源（p1 或 p4），占用资源执行的处理（p2 或 p5），不占用资源执行的处理（p3 或 p6），另外系统有一个状态：资源空闲（p7）。在有的状态中有一个黑点“□”，称为标记或令牌，表明系统或对象当前正处于此状态。当作为一个事件的输入的所有状态都得到或保有令牌时，才能引起该事件“激发”。使得系统和对象的状态向前推移，完成系统和对象的某些行为。

§ 控制规格说明

控制规格说明从两个方面给出系统的行为。其一是状态—迁移图(STD)，它是行为的“顺序规格说明”。其二是加工激活表(PAT)，它是行为的“组合规格说明”，表明当事件激发时，数据流图中的哪些加工要被激活。

控制规格说明仅描述了系统的行为，不提供被激活的加工的内部工作的细节。

(4) 数据词典

分析模型中包含了对数据对象、功能和控制的表示。在每一种表示中，数据对象和控制项都扮演一定的角色。为表示每个数据对象和控制项的特性，建立了数据词典。

数据词典精确地、严格地定义了每一个与系统相关的数据元素，并以字典式顺序将它们组织起来，使得用户和分析员对所有的输入、输出、存储成分和中间计算有共同的理解。 • 词条描述

在数据词典的每一个词条中应包含以下信息：

① 名称：数据对象或控制项、数据存储或外部实体的名字。

② 别名或编号。

③ 分类：数据对象？加工？数据流？数据文件？外部实体？控制项（事件／状态）？

④ 描述：描述内容或数据结构等。

⑤ 何处使用：使用该词条（数据或控制项）的加工。

• 内容描述

在数据词典的编制中，分析员最常用的描述内容或数据结构的符号如表 2.1 所示。

表 2.1 数据词典定义式中的符号

符 号	含 义	解 释
=	被定义为	
+	与	例如，x=a+b，表示 x 由 a 和 b 组成。
[...,...]	或	例如，x=[a,b]，x=[a|b]，表示 x 由 a 或由 b 组成。
[...|...]	或	
{...}	重复	例如，x={a}，表示 x 由 0 个或多个 a 组成。
m{...}n	重复	例如，x=3{a}8，表示 x 中至少出现 3 次 a，至多出现 8 次 a。
(...)	可选	例如，x=(a)，表示 a 可在 x 中出现,也可不出现。
"..."	基本数据元素	例如，x="a"，表示 x 为取值为 a 的数据元素。
..	连结符	例如，x=1..9，表示 x 可取 1 到 9 之中的任一值。



图 2.19 存折格式

在图 2.7 表示的取款数据流图中，数据文件“存折”的格式如图 2.19 所示，它在数据词典中的定义格式为：

存折＝户名＋所号＋帐号＋开户日＋性质＋（印密）＋1{存取行}50

户名＝2{字母}24

所号＝“001”..“999” 注：储蓄所编码，规定三位数字

帐号＝“00000001”..“99999999” 注：帐号规定由八位数字组成

开户日＝年＋月＋日

性质＝“1”..“6” 注：“1”表示普通户，“5”表示工资户等

印密＝“0” 注：印密在存折上不显示

存取行＝日期＋（摘要）＋支出＋存入＋余额＋操作＋复核

日期＝年＋月＋日

……

字母＝[“a”..“z”｜“A”..“Z”]

数据词典明确地定义了各种信息项。随着系统规模的增大，数据词典的规模和复杂性将迅速增加。

6. 用于支持需求分析的快速原型化方法

通常，原型是指模拟某种产品的原始模型。在软件开发中，原型是软件的一个早期可运行的版本，它反映最终系统的部分重要特性。如果在获得一组基本需求说明后，通过快速分析构造出一个小型的软件系统，满足用户的基本要求。使得用户可在试用原型系统的过程中得到亲身感受和受到启发，做出反应和评价。然后开发者根据用户的意见对原型加以改进。随着不断试验、纠错、使用、评价和修改，获得新的原型版本，如此周而复始，逐步减少分析和通信中的误解，弥补不足之处，进一步确定各种需求细节，适应需求的变更，从而提高了最终产品的质量。

(1) 原型的分类

由于运用原型的目的和方式不同，原型可分为以下两种不同的类型：

① 废弃型：先构造一个功能简单而且质量要求不高的模型系统，针对这个模型系统反复进行分析修改，形成比较好的设计思想，据此设计出更加完整、准确、一致、可靠的最终系统。系统构造完成后，原来的模型系统就被废弃不用。

② 追加型或演化型：先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心，然后通过不断地扩充修改，逐步追加新要求，最后发展成为最终系统。

有人把废弃型原型又细分为探索型和实验型。探索型原型的目的是要弄清对目标系统的要求，确定所希望的特性，并探讨多种方案的可行性。它主要针对开发目标模糊，用户和开发者对项目都缺乏经验的情况。而实验型原型用于大规模开发和实现之前，考核方案是否合适，规格说明是否可靠。

(2) 原型类型的选择

1984 年 Boar 提出一系列选择原型化方法的因素。如果是在需求分析阶段要使用原型化方法，必须从系统结构、逻辑结构、用户特征、应用约束、项目管理和项目环境等多方面来考虑，以决定是否采用原型化方法。

• 系统结构：联机事务处理系统，相互关联的应用系统适合于用原型化方法，而批处理、批修改等结构不适宜用原型化方法。

• 逻辑结构：有结构的系统，如操作支持系统、管理信息系统、记录管理系统等适合于用原型化方法，而基于大量算法的系统不适宜用原型化方法。

• 用户特征：不满足于预先做系统定义说明，愿意为定义和修改原型投资，不易肯定详细需求，愿意承担决策的责任，准备积极参与的用户是适合于使用原型的用户。

• 应用约束：对已经运行系统的补充，不能用原型化方法。

• 项目管理：只有项目负责人愿意使用原型化方法，才适合于用原型化的方法。

• 项目环境：需求说明技术应当根据每个项目的实际环境来选择。

当系统规模很大、要求复杂、系统服务不清晰时，在需求分析阶段先开发一个系统原型是很值得的。特别是当性能要求比较高时，在系统原型上先做一些试验也是很必要的。

1992 年 Andriole 给出 6 个问题，用来帮助选择原型方法。表 2.2 指明了对这些问题的典型答案和对使用原型方法的建议。

表 2.2 选择适当的原型方法

问题	废弃型原型法	演化型原型法	其它预备工作
目标系统要解决的问题弄清楚了吗？	是	是	否
问题可以被建模吗？	是	是	否
客户能够确定基本需求吗？	是/否	是/否	否
需求已经被建立而且比较稳定了吗？	否	是	是
有模糊不清的需求吗？	是	否	是
需求中有矛盾吗？	是	否	是

(3) 原型生存期

原型的开发和使用过程叫做原型生存期。图 2.20（a）是原型生存期的模型，图 2.20（b）是模型的细化。

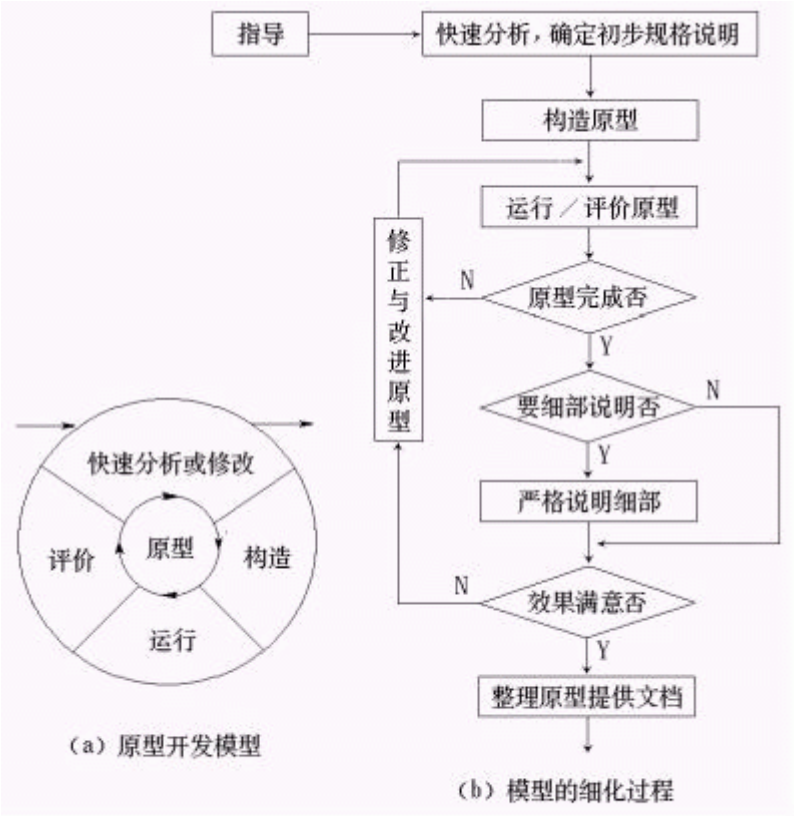


图 2.20 原型生存期

① 快速分析 ：在分析者和用户的紧密配合下，快速确定软件系统的基本要求。

② 构造原型 ：在快速分析基础上，根据基本需求，尽快实现一个可运行的系统。

③ 运行和评价原型 ：用户在开发者指导下试用原型，在试用的过程中考核评价原型的特性，分析其运行结果是否满足规格说明的要求，以及规格说明描述是否满足用户愿望。

④ 修正和改进 ： 根据修改意见进行修改。

如果用修改原型的过程代替快速分析，就形成了原型开发的迭代过程。开发者和用户在一次次的迭代过程中不断将原型完善，以接近系统的最终要求。

⑤ 判定原型完成 ：经过修改或改进的原型，达到参与者一致认可，则原型开发的迭代过程可以结束。为此，应判断有关应用的实质是否已经掌握，迭代周期是否可以结束等。

判定的结果有两个不同的转向，一是继续迭代验证，一是进行详细说明。

⑥ 判断原型细部是否说明 ：判断组成原型的细部是否需要严格地加以说明。原型化方法允许对系统必要成分或不能通过模型进行说明的成分进行严格的详细的说明。

⑦ 原型细部的说明 ：对于那些不能通过原型说明的所有项目，仍需通过文件加以说明。严格说明的成分要作为原型化方法的模型编入词典。

⑧ 判定原型效果 ：考察用户新加入的需求信息和细部说明信息，看其对模型效果有什么影响？是否会影响模块的有效性？如果模型效果受到影响，甚至导致模型失效，则要进行修正和改进。

⑨ 整理原型和提供文档。

总之，利用原型化技术，可为软件的开发提供一种完整的、灵活的、近似动态的规格说明方法。

(4) 原型开发技术

通常用于构造原型的一些技术包括可执行规格说明、基于场景的设计、自动程序设计、专用语言、可复用的软件构件和简化假设等等。其中前三种还适用于用户界面的设计。

§ 可执行规格说明 ：可执行规格说明是用于需求规格说明的一种自动化技术。可执行规格说明语言可描述系统要“做什么”，但它并不描述系统要“怎样做”。使用这种方法，人们可以直接观察他们用语言规定的任何系统性行为。可执行规格说明包括形式化规格说明、有限状态模型和可执行的数据流图。

§ 基于场景的设计 ：一个场景可模拟在系统运行期间用户经历的事件。它提供了输入—处理—输出的屏幕格式和有关对话的模型。因此，场景能够给用户显示系统的逼真的视图，使用户得以判断是否符合他的意图。

§ 自动程序设计 ：自动程序设计是可执行规格说明的替身，主要是指在程序自动生成环境的支持下，利用计算机实现软件的开发。它可以自动地或半自动地把用户的非过程性问题规格说明转换为某种高级语言程序，主要手段有以下 4 种：

§ 专用语言 ：专用语言是应用领域的模型化语言。在原型开发中使用专用语言，可方便用户和软件开发者在计划中的系统特性方面的交流。

§ 软件复用技术 ：软件复用技术可分为两大类：合成技术和生成技术。

① 合成技术：可复用的软件构件可以是对某一函数、过程、子程序、数据类型、算法等可复用软件成份的抽象，利用这些构件来构造软件系统。用构件合成较大的构件有三种方式：一是连接；二是消息传递和继承；三是管道(pipe)机制。

② 生成技术：利用可复用的模式，通过生成程序产生一个新的程序或程序段，产生的程序可以看做是模式的实例。可复用的模式有两种不同的形式：代码模式和规则模式。前者的例子是应用生成器，可复用的代码模式就存在于生成器自身。通过特定的参数替换，生成抽象软件模块的具体实体。后者的例子是变换系统，它通常采用超高级的规格说明语言，形式化地给出软件的需求规格说明，利用程

序变换系统（有时要经过一系列的变换），把用超高级规格说明语言编写的程序转化成某种可执行语言的程序。

§ 简化假设 ：简化假设是在开发过程中使设计者迅速得到一个简化的系统所做的假设。尽管这些假设可能实际上并不能成立，但它们在原型开发过程中可以使开发者的注意力集中在一些主要的方面。

7. 软件需求规格说明和需求评审

(1) 制定软件需求规格说明的原则

1979 年由 Balzer 和 Goldman 提出了做出良好规格说明的 8 条原则。

原则 1：功能与实现分离，即描述要“做什么”而不是“怎样实现”

原则 2：要求使用面向处理的规格说明语言，讨论来自环境的各种刺激可能导致系统做出什么样的功能性反应，来定义一个行为模型，从而得到“做什么”的规格说明。

原则 3：如果目标软件只是一个大系统中的一个元素，那么整个大系统也包括在规格说明的描述之中。描述该目标软件与系统的其它系统元素交互的方式。

原则 4：规格说明必须包括系统运行的环境。

原则 5：系统规格说明必须是一个认识的模型，而不是设计或实现的模型。

原则 6：规格说明必须是可操作的。规格说明必须是充分完全和形式的，以便能够利用它决定对于任意给定的测试用例，已提出的实现方案是否都能满足规格说明。

原则 7：规格说明必须容许不完备性并允许扩充。

原则 8：规格说明必须局部化和松散的耦合。它所包括的信息必须局部化，这样当信息被修改时，只要修改某个单个的段落（理想情况）。同时，规格说明应被松散地构造（即耦合），以便能够很容易地加入和删去一些段落。

尽管 Balzer 和 Goldman 提出的这 8 条原则主要用于基于形式化规格说明语言之上的需求定义的完备性，但这些原则对于其它各种形式的规格说明都适用。当然要结合实际来应用上述的原则。

(2) 软件需求规格说明

软件需求规格说明是分析任务的最终产物，通过建立完整的信息描述、详细的功能和行为描述、性能需求和设计约束的说明、合适的验收标准，给出对目标软件的各种需求。

表 2.3 软件需求规格说明的框架

I. 引言 A.系统参考文献 B.整体描述 C.软件项目约束
II. 信息描述 A.信息内容表示 B.信息流表示 i 数据流 ii 控制流
III. 功能描述 A.功能划分 B.功能描述 i 处理说明 ii 限制/局限 iii. 性能需求 iv. 设计约束 v. 支撑图 C. 控制描述 i 控制规格说明 ii. 设计约束
IV. 行为描述 A.系统状态 B.事件和响应
V. 检验标准 A.性能范围 B.测试种类 C.期望的软件响应 D.特殊的考虑
VI. 参考书目
VII. 附录

(3) 需求规格说明评审

作为需求分析阶段工作的复查手段，在需求分析的最后一步，应该对功能的正确性、完整性和清晰性，以及其它需求给予评价。评审的主要内容是：

- 系统定义的目标是否与用户的要求一致；
- 系统需求分析阶段提供的文档资料是否齐全；
- 文档中的所有描述是否完整、清晰、准确反映用户要求；
- 与所有其它系统成分的重要接口是否都已经描述；
- 被开发项目的数据流与数据结构是否足够，确定；
- 所有图表是否清楚，在不补充说明时能否理解；
- 主要功能是否已包括在规定的软件范围之内，是否都已充分说明；
- 软件的行为和它必须处理的信息、必须完成的功能是否一致；
- 设计的约束条件或限制条件是否符合实际；
- 是否考虑了开发的技术风险；
- 是否考虑过软件需求的其它方案；
- 是否考虑过将来可能会提出的软件需求；
- 是否详细制定了检验标准，它们能否对系统定义是否成功进行确认；
- 有没有遗漏，重复或不一致的地方；
- 用户是否审查了初步的用户手册或原型；
- 软件开发计划中的估算是否受到了影响。

为保证软件需求定义的质量，评审应以专门指定的人员负责，并按规程严格进行。评审结束应有评审负责人的结论意见及签字。除分析员之外，用户／需求者，开发部门的管理者，软件设计、实现、测试的人员都应当参加评审工作。一般，评审的结果都包括了一些修改意见，待修改完成后再经评审通过，才可进入设计阶段。

三、例题分析

【例 1】软件需求分析阶段的工作，可以分为以下 4 个方面：对问题的识别、分析与综合、编写需求分析文档以及( )。

供选择的答案：

A. 总结 B. 阶段性报告 C. 需求分析评审 D. 以上答案都不正确

答案： C.

分析：作为需求分析阶段工作的复查手段，在需求分析的最后一步，应该对功能的正确性、完整性和清晰性，以及其它需求给予评价。一般，评审的结果都包括了一些修改意见，待修改完成后再经评审通过，才可进入设计阶段。

【例 2】各种需求方法都有它们共同适用的( )。

供选择的答案：

A. 说明方法 B. 描述方式 C. 准则 D. 基本原则

答案： D.

分析：虽然各种分析方法都有独特的描述方法，但所有的分析方法还是有它们共同适用的基本原则。这些基本原则包括：

§ 需要能够表达和理解问题的信息域和功能域；

§ 要能以层次化的方式对问题进行分解和不断细化；

§ 要分别给出系统的逻辑视图和物理视图。

【例 3】在结构化分析方法中，用以表达系统内数据的运动情况的工具有( )。

供选择的答案：

A. 数据流图 B. 数据词典 C. 结构化英语 D. 判定表与判定树

答案： A.

分析：数据流图从数据传递和加工的角度，以图形的方式刻画数据流从输入到输出的移动变换过程，所以，它不是描述数据的静态结构，而是描述数据流的传递和变换。数据词典主要用于定义数据和控制对象的细节，结构化英语、判定表和判定树主要用于描述加工规格说明，都不是表达数据在系统内运动情况的工具。

【例 4】在结构化分析方法中用状态－迁移图表达系统或对象的行为。在状态－迁移图中，由一个状态和一个事件所决定的下一状态可能会有( )个。

供选择的答案：

A. 1 B. 2 C. 多个 D. 不确定

答案： C.

分析：在状态－迁移图中，由一个状态和一个事件所确定的下一状态可能会有多个。实际会迁移到哪一个状态，是由更详细的内部状态和更详细的事件信息来决定的，此时在状态－迁移图中可能需要使用加进判断框和处理框的记法。状态－迁移图的优点：第一，状态之间的关系能够直观地捕捉到，这样用眼睛就能看到是否所有可能的状态迁移都已纳入图中，是否存在不必要的状态等。第二，由于状态－迁移图的单纯性，能够机械地分析许多情况，可很容易地建立分析工具。

【例 5】在结构化分析方法中用实体－关系图表达系统中的对象及其关系。在实体－关系图中，表达对象的实例之间的关联有三种类型：一对一联系、( )联系、多对多联系。

供选择的答案：

A. 多对一 B. 一对多

分析：使用实体－关系图，可以建立系统中各个数据对象及对象之间的关系。对象的实例间的关联称为“基数”，共有 3 种类型的基数：一对一，一对多，多对多。它反映了现实世界中实体之间的联系，多对一的情况可以归入一对多的关联中去。

【例 6】 软件需求分析的任务不应包括(A)。进行需求分析可使用多种工具，但(B)是不适用的。在需求分析中，分析员要从用户那里解决的最重要的问题是(C)。需求规格说明书的内容不应当包括(D)。该文档在软件开发中具有重要的作用，但其作用不应当包括(E)。

供选择的答案：

A. ① 问题分析 ② 信息域分析 ③ 结构化程序设计 ④ 确定逻辑模型

B. ① 数据流图 ② 判定表 ③ PAD 图 ④ 数据词典

C. ① 要让软件做什么 ② 要给该软件提供哪些信息

③ 要求软件工作效率如何 ④ 要让软件具有什么样的结构

D. ① 对重要功能的描述 ② 对算法的详细过程性描述

③ 软件确认准则 ④ 软件的性能

E. ① 软件设计的依据 ② 用户和开发人员对软件要“做什么”的共同理解

③ 软件验收的依据 ④ 软件可行性分析的依据 答案：A. ③, B. ③, C. ①, D. ②, E. ④

分析：软件需求分析的任务是通过与用户的合作，了解用户对待开发系统的要求；根据对用户要求的系统所在的信息域的调查、分析，确定系统的逻辑模型；并对求解的问题做适当的分解，使之适合于计算机求解。需求分析的结果是软件需求规格说明书。

结构化程序设计是在详细设计和编码阶段所采用的技术，而不是需求分析阶段要采用的技术。在需求分析阶段，分析人员可以用数据流图描述系统的数据流的变换和流向，用数据词典定义在数据流图中出现的数据流、数据文件、加工或处理，用判定表表示复杂条件和动作组合的情况。但 PAD 图是在详细设计阶段使用的描述加工逻辑的工具，不适用于需求分析。此外，软件需求分析阶段只确定软件系统要“做什么”，完成对重要功能、性能、确认准则的描述，至于“怎么做”由后续的设计阶段完成，对算法的详细过程性描述也是在设计阶段给出。软件可行性分析应在需求分析之前，所以需求分析规格说明不能成为可行性分析的依据。

【例 7】原型化方法是用户和软件开发人员之间进行的一种交互过程，适用于(A)系统。它从用户界面的开发入手，首先形成(B)，用户(C)，并就(D)提出意见，它是一种(E)型的设计过程。

供选择的答案：

A. ① 需求不确定性高的 ② 需求确定的 ③ 管理信息 ④ 决策支持

B. ① 用户界面使用手册 ② 用户界面需求分析说明书

③ 系统界面原型 ④ 完善的用户界面

C. ① 改进用户界面的设计 ② 阅读文档资料

③ 模拟用户界面的运行 ④ 运行用户界面原型

D.① 同意什么和不同意什么 ② 使用和不使用哪一种编程语言

③ 程序的结构 ④ 执行速度是否满足要求

E.① 自外向内 ② 自顶向下 ③ 自内向外 ④ 自底向上

答案：A. ① B. ③ C. ④ D. ① E. ①

分析：通常，原型是指模拟某种产品的原始模型。在软件开发中，原型是软件的一个早期可运行的版本，它反映最终系统的部分重要特性。

使用原型的原型化方法特别适用于需求不确定性较高的软件系统的开发。它的基本思想是根据用户给出的基本需求，通过快速实现构造出一个小型的可执行的模型，满足用户的基本要求，这就是系统界面原型。让用户计算机上实际运行这个用户界面原型，在试用的过程中得到亲身感受和受到启发，做出反应和评价，提出同意什么和不同意什么。然后开发者根据用户的意见对原型加以改进。随着不断试验、纠错、使用、评价和修改，获得新的原型版本，如此周而复始，逐步减少分析和通信中的误解，弥补不足之处，进一步确定各种需求细节，适应需求的变更，从而提高了最终产品的质量。

它是一种自外向内型的设计过程。

四、习题

【2-1】在软件需求分析时，首先建立当前系统的物理模型，再根据物理模型建立当前系统的逻辑模型。试问：什么是当前系统？当前系统的物理模型与逻辑模型有什么差别？

【2-2】软件需求分析是软件工程过程中交换意见最频繁的步骤。为什么交换意见的途径会经常阻塞？

【2-3】你认为一个系统分析员的理想训练和基础知识是什么？请说明理由。

【2-4】可行性研究主要研究哪些问题？试说明之。

【2-5】信息和信息结构有什么区别？有没有不存在信息流的系统？有没有不存在信息结构的系统？

【2-6】软件需求分析的操作性原则和需求工程的指导性原则是什么？

【2-7】数据流图的作用是什么？它有哪些基本成份？

【2-8】考务处理系统的分层数据流图如下图所示。

该考务处理系统有如下功能：

① 对考生送来的报名表进行检查；

② 对合格的报名表编好准考证号码后将准考证送给考生，并将汇总后的考生名单送给阅卷站；

③ 对阅卷站送来的成绩表进行检查，并根据考试中心指定的合格标准审定合格者；

④ 填写考生通知单（内容包含考试成绩及合格／不合格标志），送给考生；

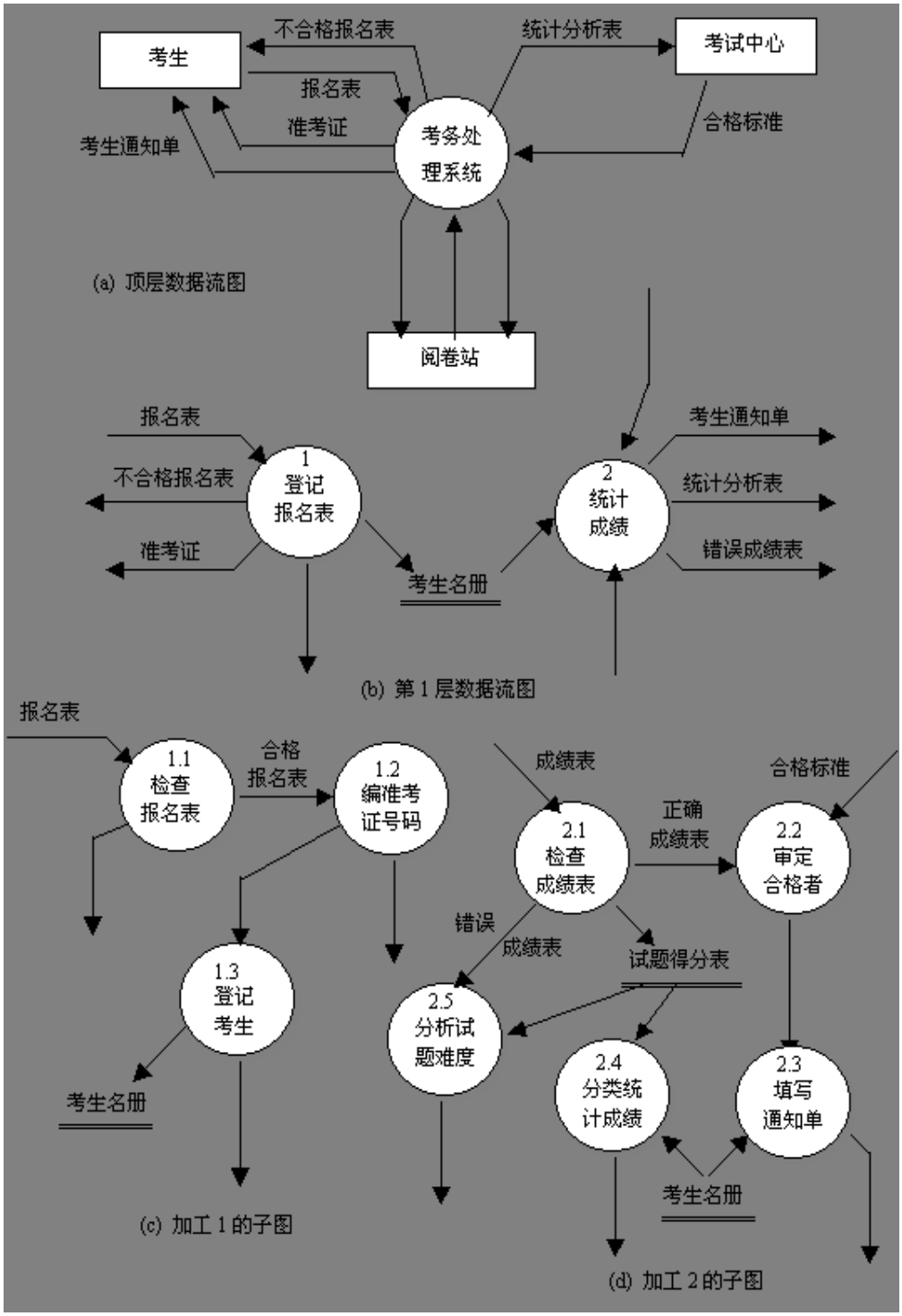
⑤ 按地区、年龄、文化程度、职业、考试级别等进行成绩分类统计及试题难度分析，产生统计分析表。

(1) 图(c)中，加工 1.1 的输入数据流是(A)，输出数据流是(B)，图(b)中，加工 2 的输出数据流是(C)，它是由(D)和(E)组成。

供选择的答案：

A ~ E. ① 统计分析表 ② 报名表 ③ 准考证 ④ 考生通知单

⑤ 合格报名表 ⑥ 难度分析表 ⑦ 错误成绩表 ⑧ 分类统计表



(2) 图(d)中的文件“试题得分表”是否在图(b)中漏掉了？ 回答是(F)。

供选择的答案：

F. ① “试题得分表”没有在图(b)中画出，是错误的。

② “试题得分表”是图(b)中加工的内部文件，不必在图(b)中画出。

③ “试题得分表”是多余的。

【2-9】Petri 网可以描述计算机软件系统的执行。现有一个程序如下（类似于 Pascal 语言）

```
L : S1;

WHILE P1 DO

BEGIN

IF P2 THEN S2

ELSE S3;

COBEGIN

S4;

S5;

S6;

COEND

END;

GOTO L;
```

其中，P1 和 P2 为逻辑表达式，S1~S6 是单个执行语句，COBEGIN 和 COEND 是并行执行开始和并行执行结束（即 S4，S5 和 S6 语句并行执行）。试用 Petri 网描述这段程序的执行过程。

【2-10】数据词典的作用是什么？它有哪些基本词条？

【2-11】传统的软件开发模型的缺陷是什么？原型化方法的类型有哪些？原型开发模型的主要优点是什么？

【2-12】试简述原型开发的过程和运用原型化方法的软件开发过程。

【2-13】软件需求分析说明书主要包括哪些内容?

【2-14】阅读下列关于开发人事管理系统的交互式工作方式的叙述，再回答问题。

某大企业最近决定采用高性能微机开发人事管理系统，将四台联机终端分置于人事处的三个科室。该系统可供操作员和程序员使用，也可供人事处负责人和主管

人事的副厂长等查询人事信息用。人事管理系统通过录入人事数据和修改、删除等操作，产生和更新各类人事文件，通过搜索这些文件进行各类人事信息的查询。该企业有 3000 多个工人、干部和技术人员，大体可分成机关科室、生产车间、后勤服务和开发研制部门等几类部门。厂领导决定由计算机应用科来负责协调和开发应用系统。计算机应用科科长指示系统工程师张某负责进行系统分析。

考虑到人事处有大量的查询信息要求、频繁的人事信息修改和文件存档、查阅等特点，计算机应用科决定认真设计人机交互界面，首先设计好在终端上的交互式会话的方式。

系统工程师张某通过调查收集到如下 10 条意见：

(1) 某程序员认为：系统在屏幕格式、编码等方面应具有一致性和清晰性，否则会影响操作人员的工作效率。

(2) 某操作人员认为：在交互式会话过程中，操作人员可能会忘记或记错某些事情，系统应当提供 HELP 功能。

(3) 某操作人员认为：既然是交互式会话，那么对所有的输入都应当作出响应，不应出现击键后，计算机没有任何反应的情况。

(4) 某操作人员认为：在出错的时候，交互式会话系统应当给出出错信息，并且尽可能告诉我们出错的性质和错在什么地方。

(5) 某程序员认为：终端会话也应当符合程序员编制程序时的习惯，这样可以更高效地维护人事管理系统。

(6) 教育科干部甲认为：应当对操作员进行一些必要的培训，让他们掌握交互式会话系统的设计技巧，有助于提高系统的使用效率。

(7) 教育科干部乙认为：尽管操作人员的指法已经强化训练但在交互式会话时应尽可能缩短和减少操作员输入的信息，以降低出错概率。

(8) 某程序员认为：由于本企业中有很多较大的文件，文件的查找很费时间，交互式会话系统在响应时间较长时应给予使用者以提示信息。

(9) 人事处干部丙认为：我们企业的人事资料相当复杂，格式非常之多，希望交互式系统使用十分清晰的格式，并容易对输入数据中的错误进行修改。

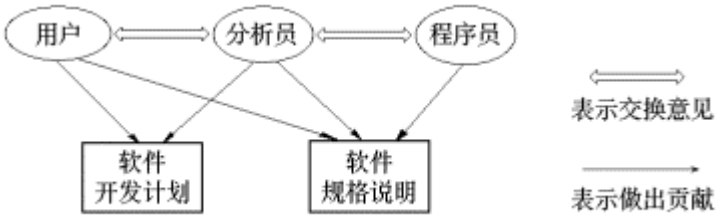
(10) 人事处干部丁认为：人事管理系统应当具有相当的保密性和数据安全性，因此在屏幕上显示出的信息应该含混一些，以免泄密。 系统工程师张某对上述调查情况和其他要求作了分析后，发现收集到的 10 条意见中有 3 条意见是不能接受的，写出编号并各用 40 字以内叙述理由。

五、习题解答

【2-1】所谓当前系统可能是需要改进的某个已在计算机上运行的数据处理系统，也可能是一个人工的数据处理过程。当前系统的物理模型客观地反映当前系统实际的工作情况。但在物理模型中有许多物理的因素，随着分析工作的深入，有些非本质的物理因素就成为不必要的负担，因而需要对物理模型进行分析，区分出本质的和非本质的因素，去掉那些非本质的因素即可获得反映系统本质的逻辑模型。所以当前系统的逻辑模型是从当前系统的物理模型抽象出来的。

【2-2】软件需求分析过程中，由于最初分析员对要解决的问题了解很少，用户对问题的描述、对目标软件的要求也很凌乱、模糊，再加上分析员和用户共同的知识领域不多，导致相互间通信的需求。首先，由于分析员和用户之间需要通信的内容相当多，业务知识上的不足，表达方式的不足，可能对某些需求存在错误解释或误解的可能性，造成需求的模糊性。其次，用户和分析员之间经常存在无意识的“我们和他们”的界限，不是按工作需要组成统一的精干的队伍，而是各自定义自己的“版图”，并通过一系列备忘录、正式的意见书、文档，以及提问和回答来相互通信。历史已经证明，这样会产生大量误解。忽略重要信息，无法建立成功的工作关系。

【2-3】系统分析员处在用户和高级程序员之间，负责沟通用户和开发人员的认识和见解，起着桥梁的作用。一方面要协助用户对所开发的软件阐明要求，另一方面还要与高级程序员交换意见，探讨用户所提要求的合理性以及实现的可能性。最后还要负责编写软件需求规格说明和初步的用户手册。



为能胜任上述任务，分析员应当具备如下的素质:

(1) 能够熟练地掌握计算机硬、软件的专业知识，具有一定的系统开发经验。

(2) 善于进行抽象的思维和创造性的思维，善于把握抽象的概念，并把它们重新整理成为各种逻辑成分，并给出简明、清晰的描述。

(3) 善于从相互冲突或混淆的原始资料中抽出恰当的条目来。

(4) 善于进行调查研究，能够很快学习用户的专业领域知识，理解用户的环境条件。

(5) 能够倾听他人的意见，注意发挥其它人员的作用。

(6) 具有良好的书面和口头交流表达能力。

【2-4】可行性研究主要做 4 个方面的研究：

§ 经济可行性 ：进行成本 / 效益分析。从经济角度判断系统开发是否“合算”。

§ 技术可行性 ：进行技术风险评价。从开发者的技术实力、以往工作基础、问题的复杂性等出发，判断系统开发在时间、费用等限制条件下成功的可能性。

§ 法律可行性 ：确定系统开发可能导致的任何侵权、妨碍和责任。

§ 方案的选择 ：评价系统或产品开发的几个可能的候选方案。最后给出结论意见。

【2-5】什么是信息？广义地讲，信息就是消息。宇宙三要素（物质、能量、信息）之一。它是现实世界各种事物在人们头脑中的反映。此外，人们通过科学仪器能够认识到的也是信息。信息的特征为：可识别、可存储、可变换、可处理、可传递、可再生、可压缩、可利用、可共享。我们通常讲的信息域就是对信息的多视角考虑。信息域包含 3 个不同的视图：信息内容和关系、信息流和信息结构。为了完全理解信息域，必须了解每一个视图。

信息结构：它是信息在计算机中的组织形式。一般表示了各种数据和控制对象的内部组织。数据和控制对象是被组织成 n 维表格，还是组织成有层次的树型结构？在结构中信息与其它哪些信息相关？所有信息是在一个信息结构中，还是在几个信息结构中？一个结构中的信息与其它结构中的信息如何联系？这些问题都由信息结构的分析来解决。 信息流：表示数据和控制在系统中传递时的变化方式。输入对象首先被变换成中间信息（数据或控制），然后再变换成输出结果信息。沿着变换路径，可能从已有的数据存储（如磁盘文件或内存缓冲区）中引入附加的信息。对数据进行变换是程序中应有的功能或子功能。两个变换功能之间的数据传递就确定了功能间的接口。

所以，没有信息流的系统相当于没有功能的系统，这样的系统的存在是毫无意义的。而没有信息结构的系统是没有信息的系统，这样的系统不是计算机能够处理的系统。

【2-6】所有的需求分析方法都与一组操作性原则相关联：

§ 必须理解和表示问题的信息域。

§ 必须定义软件将完成的功能。

§ 必须表示软件的行为（作为外部事件的结果）。

§ 必须对描述信息、功能和行为的模型进行分解，能够以层次方式揭示其细节。

§ 分析过程应当从要素信息转向细节的实现。

通过使用这些原则，分析员可以系统地处理问题。首先检查信息域以便更完整地理解目标软件的功能，再使用模型以简洁的方式表达目标软件的功能和行为，并利用自顶向下、逐层分解的手段来降低问题的复杂性。在这些处理过程中，因处理需求带来的逻辑约束和因其它系统元素带来的物理约束需要通过软件要素和视图的实现加以检验和确认。 除此以外，Davis 建议了一组针对“需求工程”的指导性原则：

§ 在开始建立分析模型之前应当先理解问题。如果问题没有很好理解就急于求成，常常会产生一个解决错误问题的完美的软件。

§ 强力推荐使用原型。这样做可以使用户了解将如何与计算机交互，而人们对软件质量的认识常常是基于对界面“友好性”的切身体会。

§ 记录每一个需求的起源和原因。这是建立对用户要求的可追溯性的第一步。

§ 使用多个视图，建立系统的数据、功能和行为模型。这样做可帮助分析员从多方面分析和理解问题，减少遗漏，识别可能的不一致之处。

§ 给需求赋予优先级。因为过短的时限会减少实现所有软件需求的可能性。因此，对需求排定一个优先次序，标识哪些需求先实现，哪些需求后实现。

§ 注意消除歧义性。因为大多数需求都是以自然语言描述，存在叙述的歧义性问题，造成遗漏和误解。采用正式的技术评审是发现和消除歧义性的好方法。

遵循以上原则，就可能开发出较好的软件需求规格说明，为软件设计奠定基础。

【2-7】数据流图可以用来抽象地表示系统或软件。它从信息传递和加工的角度，以图形的方式刻画数据流从输入到输出的移动变换过程，同时可以按自顶向下、逐步分解的方法表示内容不断增加的数据流和功能细节。因此，数据流图既提供了功能建模的机制，也提供了信息流建模的机制，从而可以建立起系统或软件的功能模型。 数据流图的基本成份有 4 种：



加工。输入数据在此进行变换产生输出数据，其中要注明加工的名字。



数据输入的源点(Source)或数据输出的汇点(Sink)。其中要注明源点或汇点的名字。

数据流。被加工的数据与流向，箭头边应给出数据流名字，可用名词或名词性短语命名。



数据存储文件。也必须加以命名，用名词或名词性短语命名。

【2-8】答案：A. ② B. ⑤ C. ① D. ⑥ E. ⑧ F. ②

其中，D 与 E 的答案可互换。

应注意的问题：

① 适当地为数据流、加工、文件、数据的源／汇点命名。名字应反映该元素的实际含义，避免空洞的名字。如数据、信息处理、计算等名字都不好。 ② 画数据流时不要夹带控制流。数据流图中各种数据的加工没有考虑时序关系，引入控制流后，加工之间就有了时序关系，这与画数据流图不考虑实现细节的初衷相违背。

③ 一个加工的输出数据流不要与该加工的输入数据流重名，即使它们的组成成分相同。例如图(c)中加工 1.1 的输入数据流“报名表”与输出数据流“合格报名表”。

④ 允许一个加工有多个数据流流向另一个加工，也允许一个加工有两个相同的输出数据流流向两个不同的加工。

⑤ 保持父图与子图的平衡。就是说，父图与它的子图的输入数据流与输出数据流应当在数量与名字上都相同。特别的是，如果父图的一个输入（或输出）数据流对应于子图中几个输入（或输出）数据流，但子图中这几个数据流中的数据项合起来正好是父图中的那个数据流，这时它们还算是平衡的。例如，图(b)中加工 2 的输出数据流“统计分析表”是由“难度分析表”和“分类统计表”组成，那么图(b)与图(d)仍满足父图与子图平衡的条件。

⑥ 在自顶向下的分解过程中，若一个文件首次出现时只与一个加工有关，那么这个文件应作为这个加工的内部文件而不必画出。例如，图(d)中的文件“试题得分表”就是图(b)中加工的内部文件，所以在图(b)中没有画出。

⑦ 保持数据守恒。就是说，一个加工的所有输出数据流中的数据必须能从该加工的输入数据流中直接获得，或者是通过该加工产生的数据。

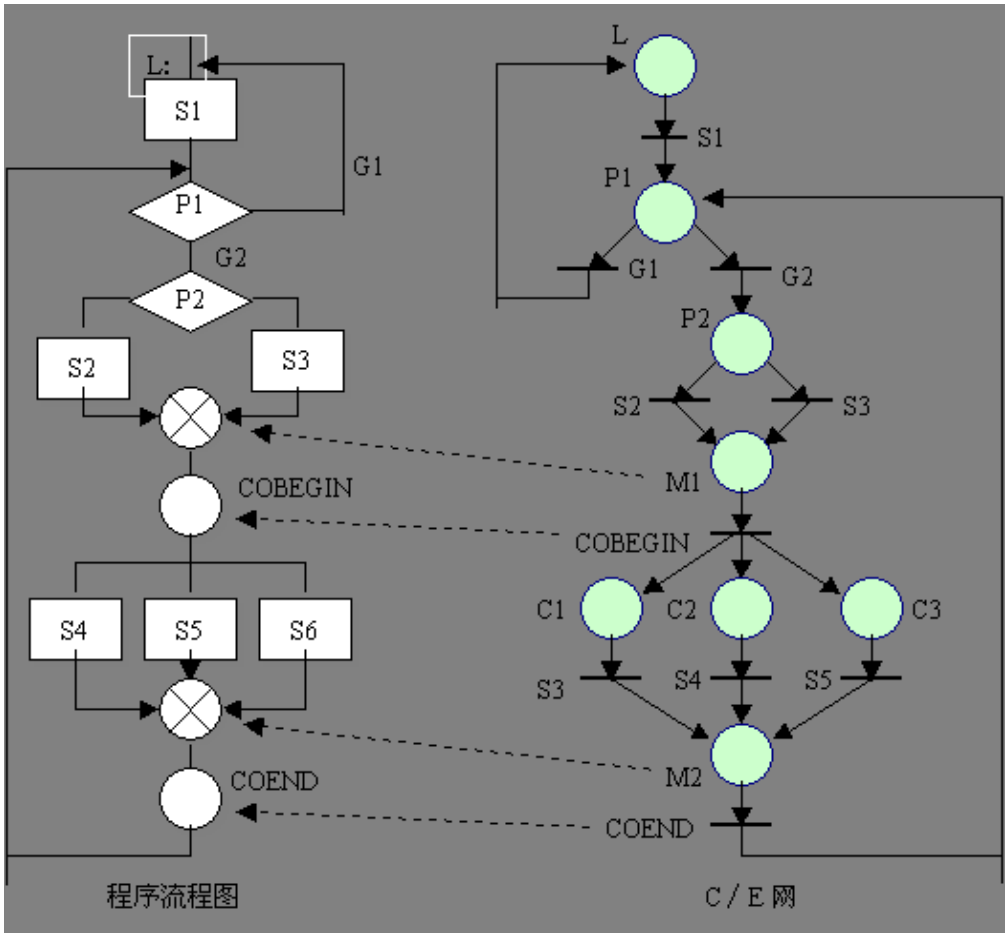
【2-9】采用条件／事件网（C／E 网，C—Condition, E—Event）式 Petri 网。其定义如下：

① 当事件 e 激发时条件 c 开始成立，则称 c 是 e 的后继。此关系用“ $\mapsto$ ”表示；

② 当事件 e 激发时条件 c 消失成立，则称 c 是 e 的前驱。此关系用“ $\dashv$ ”表示；

③ 当事件 e 激发时条件 c 不受影响，则 c 和 e 之间没有前驱、后继关系，无边。

根据定义，给定程序的 C／E 网如下：



【2-10】分析模型中包含了对数据对象、功能和控制的表示。在每一种表示中，数据对象和控制项都扮演一定的角色。为表示每个数据对象和控制项的特性，建立了数据词典。数据词典精确地、严格地定义了每一个与系统相关的数据元素，并以字典式顺序将它们组织起来，使得用户和分析员对所有的输入、输出、存储成分和中间计算有共同的理解。

在数据词典的每一个词条中应包含以下信息：

① 名称：数据对象或控制项、数据存储或外部实体的名字。

② 别名或编号。

③ 分类：数据对象？加工？数据流？数据文件？外部实体？控制项（事件／状态）？

④ 描述：描述内容或数据结构等。

⑤ 何处使用：使用该词条（数据或控制项）的加工。

【2-11】传统软件生存期范型的典型代表是“瀑布模型”。这种模型的核心是将软件生存期划分为软件计划、需求分析、软件设计、编码、测试和运行维护等阶

段，根据不同阶段工作的特点，运用不同的方法、技术和工具来完成该阶段的任务。软件开发人员遵循严格的规范，在每一阶段工作结束时都要进行严格的阶段评审和确认，以得到该阶段的一致、完整、正确和无歧义性的文档资料，并以它们做为下一阶段工作的基础。 传统思想强调每一阶段的严格性，尤其是开发初期要有良好的软件规格说明，主要是源于过去软件开发的经验教训，即在开发的后期或运行维护期间来修改不完善的规格说明要付出巨大的代价。但是，要想得到一个完整准确的规格说明不是一件容易的事。特别是对于一些大型的软件项目，在开发的早期用户往往对系统只有一个模糊的想法，很难完全准确地表达对系统的全面要求，软件开发人员对于所要解决的应用问题认识更是模糊不清。经过详细的讨论和分析，也许能得到一份较好的规格说明，但却很难期望该规格说明能将系统的各个方面都描述得完整、准确、一致，并与实际环境相符。很难通过它在逻辑上推断出（不是在实际运行中判断评价）系统运行的效果，以此达到各方对系统的共同理解。随着开发工作向前推进，用户可能会产生新的要求，或因环境变化，要求系统也能随之变化；开发人员又可能在设计与实现的过程中遇到一些没有预料到的实际困难，需要以改变需求来解脱困境。因此规格说明难以完善、需求的变更、以及通信中的模糊和误解，都会成为软件开发顺利推进的障碍。尽管在传统软件生存期管理中通过加强评审和确认，全面测试，甚至依靠维护阶段能够缓解上述问题，但不能从根本上解决这些问题。 为了解决这些问题，逐渐形成了软件系统的快速原型的概念。由于运用原型的目的和方式不同，原型可分为以下两种不同的类型：

① 废弃型：先构造一个功能简单而且质量要求不高的模型系统，针对这个模型系统反复进行分析修改，形成比较好的设计思想，据此设计出更加完整、准确、一致、可靠的最终系统。系统构造完成后，原来的模型系统就被废弃不用。

② 追加型或演化型：先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心，然后通过不断地扩充修改，逐步追加新要求，最后发展成为最终系统。

建立快速原型进行系统的分析和构造，有以下的优点：

① 增进软件者和用户对系统服务需求的理解，使比较含糊的具有不确定性的软件需求（主要是功能）明确化。由于这种方法能在早期就明确了用户的要求，因此可防止以后由于不能满足用户要求而造成的返工，从而避免了不必要的经济损失，缩短了开发周期。

② 软件原型化方法提供了一种有力的学习手段。通过原型演示，用户可以亲身体验早期的开发过程，获得关于计算机和被开发系统的专门知识。软件开发人员也可以获得用户对系统的确切要求，学习到应用范围的专业知识。

③ 使用原型化方法，可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果。因而它可以作为理解和确认软件需求规格说明的工具。

④ 软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。

【2-12】原型的开发和使用过程叫做原型生存期。下图是原型生存期的模型及其细化。

① 快速分析 ：在分析者和用户的紧密配合下，快速确定软件系统的基本要求。

② 构造原型 ：根据基本规格说明，尽快实现一个可运行的原型系统。

③ 运行和评价原型 ：用户试用原型，考核评价原型的特性。纠正过去交互中的误解和分析中的错误，增补新的要求，提出全面的修改意见。

④ 修正和改进 ：根据修改意见进行修改。如果用修改原型的过程代替快速分析，就形成了原型开发的迭代过程。在一次次的迭代过程中不断将原型完善，以接近系统的最终要求。 ⑤ 判定原型完成 ：经过修改或改进的原型，达到参与者一致认可，则原型开发的迭代过程可以结束。为此，应判断有关应用的实质是否已经掌握，判定的结果有两个不同的转向，一是继续迭代验证，一是进行详细说明。

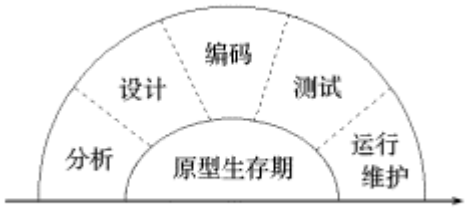
⑥ 判断原型细部是否说明 ：判断组成原型的细部是否需要严格地加以说明。

⑦ 原型细部的说明 ：通过文件加以说明那些不能通过原型说明的项目。

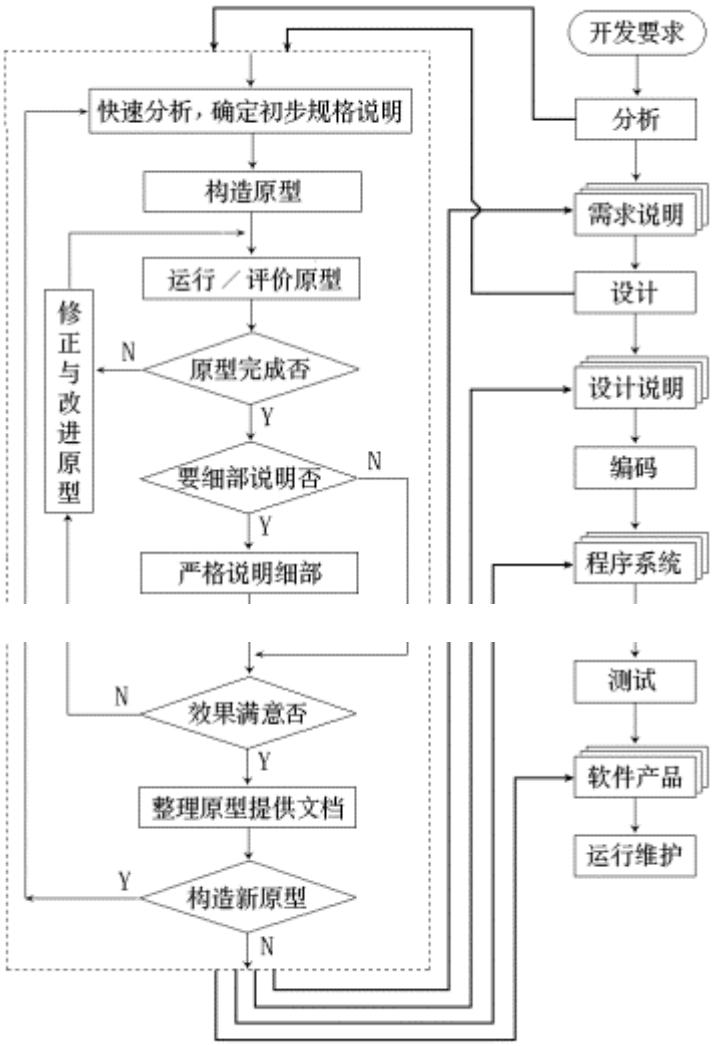
⑧ 判定原型效果 ：考察新加入的需求信息和细部说明信息，看其对模型有什么影响？是否会影响模块的有效性？如果模型受到影响，则要进行修正和改进。

⑨ 整理原型和提供文档

快速原型方法的提出使得传统的软件生存期在思想方法上受到了影响。如果只是在局部运用原型化方法，若将原型开发过程用于软件生存期的某一个阶段内，那么传统的软件生存期依然不变，只是阶段内部的软件定义或开发活动采用了新的方法。但若原型开发过程代替了传统生存期中的多个阶段，则软件开发过程就成为一种新的形式。



（a） 软件开发模型



（b） 模型的细化

图(a)表示了使用原型方法的软件生存期模型。原型开发过程处于核心，表示可在生存期的任何阶段中引入原型开发过程，也可合并若干阶段，用原型开发过程代替。图(b)详细描述了在各个阶段可能引入原型开发过程的软件开发过程。其中在原型开发过程的最后加上了一个“是否构造新原型”的判断，这是针对在系统开发的过程中有可能为不同的目的而要使用多个原型的情况而设。

① 辅助或代替分析阶段 ：在分析阶段利用快速原型方法可以得到良好的需求规格说明。在整体上仍然采用传统的模式，但使用原型化方法来补充和完善需求说明以达到一致、准确、完整、无多义性地反映用户要求，从而代替了传统的仅由复审和确认来提高需求规格说明质量的方法。并能在早期克服潜在的误解、遗漏和错误，尽量不让潜在的问题遗留到开发的后期，减少将来维护的代价。

② 辅助设计阶段 ：在设计阶段引入原型，可根据需求分析得到的规格说明进行快速分析，得到实现方案后立即构造原型，通过运行，考察设计方案的可行性与

合理性。在这个阶段引入原型，可以迅速得到完善的设计规格说明。原型可能成为设计的总体框架，也可能成为最终设计的一部分或补充的设计文档。

③ 代替分析与设计阶段 ：这时不再遵循传统的严格按阶段进行软件开发的要求，而是把原型方法直接应用到软件开发的整体过程。在实施原型开发的过程中，不再考虑完善的需求说明，把分析、定义和设计交织在一起，通过原型的构造、评价与改进的迭代过程，逐步向最终系统的全面要求靠近。由于在分析的同时也考虑了设计与实现的要求，能更有效地确定系统的需求和设计规格说明。

④ 代替分析、设计和实现阶段 ：在软件开发环境的支持下，通过原型生存期的反复迭代，直接得到软件的程序，交付系统测试。这属于进化型的原型开发，由初始的基本需求得到最初的原型开始，一直进化到软件的整体系统，并满足用户的一切可能的要求。

⑤ 代替全部定义与开发阶段 ：这是典型的进化型原型开发方法。完全摆脱了传统的软件生存期模式，通过反复的原型迭代过程，直接得到最终的软件产品。系统测试作为原型评价工作的一部分，融入原型的开发过程。

【2-13】软件需求规格说明是分析任务的最终产物，通过建立完整的信息描述、详细的功能和行为描述、性能需求和设计约束的说明、合适的验收标准，给出对目标软件的各种需求。

软件需求规格说明的框架如下：

Ⅰ. 引言 A.系统参考文献 B.整体描述 C.软件项目约束

Ⅱ. 信息描述 A.信息内容表示 B.信息流表示 i 数据流 ii控制流

Ⅲ. 功能描述 A.功能划分 B.功能描述 i 处理说明 ii限制/局限 iii 性能需求

iv 设计约束 v 支撑图 C.控制描述 i 控制规格说明 ii 设计约束

Ⅳ. 行为描述 A.系统状态 B.事件和响应

Ⅴ. 检验标准 A.性能范围 B.测试种类 C.期望的软件响应 D.特殊的考虑

Ⅵ. 参考书目

Ⅶ. 附录

【2-14】不能接受的 3 条意见是 (5)、(6)、(10)。人机交互界面首先考虑的是用户如何使用起来方便，与编程习惯、设计技巧无关。此外，屏幕上信息应很清晰易懂，安全保密与屏幕显示无关。

第三章 软件设计

一、复习要求

1. 了解软件概要设计的原则和过程。
2. 掌握模块划分的评价准则－模块独立性的判别。
3. 掌握结构化设计方法。
4. 了解 Jackson 系统开发方法和 Jackson 程序设计方法。
5. 了解数据设计和文件设计的原则。
6. 掌握常用的详细设计的表达方法。
7. 了解软件设计规格说明和设计评审的主要内容。

二、内容提要

1. 软件设计的过程

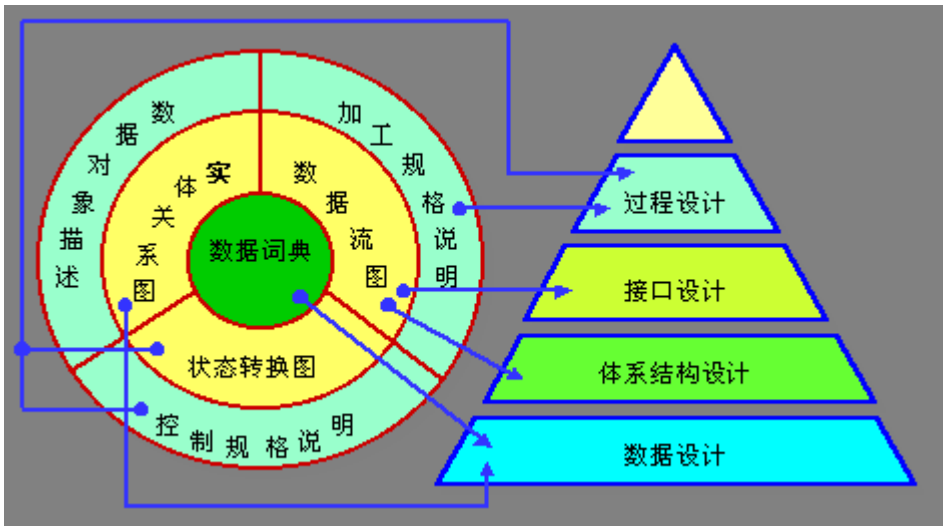
一旦软件需求确定之后，就进入开发阶段。开发阶段由三个互相关联的的步骤组成：设计、实现（编码）和测试。每个步骤都按某种方式进行信息变换，最后得到有效的计算机软件。

(1) 软件设计在开发阶段中的重要性

在软件需求分析阶段已经完全弄清楚了软件的各种需求，较好地解决了要让所开发的软件"做什么"的问题，并已在软件需求规格说明和数据要求规格说明中详尽和充分地阐明了这些需求。下一步就要着手实现软件的需求，即要着手解决"怎么做"的问题。

分析模型中的每一个成份都提供了建立设计模型所需的信息。软件设计的信息流如图 4.1 所示。根据用数据、功能和行为模型表示的软件需求，采用某种设计方法进行数据设计、体系结构设计、接口设计和过程设计。

图 3.1 将分析模型转换为软件设计



数据设计将实体－关系图中描述的对象和关系，以及数据词典中描述的详细数据内容转化为数据结构的定义。体系结构设计定义软件系统各主要成份之间的关系。接口设计根据数据流图定义软件内部各成份之间、软件与其它协同系统之间

及软件与用户之间的交互机制。过程设计则是把结构成份转换成软件的过程性描述。在编码步骤，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件。

软件设计是开发阶段中最重要的步骤，它是软件开发过程中质量得以保证的关键步骤。设计提供了软件的表示，使得软件的质量评价成为可能。同时，软件设计又是将用户要求准确地转化成为最终的软件产品的唯一途径。另一方面，软件设计是后续开发步骤及软件维护工作的基础。如果没有设计，只能建立一个不稳定的系统，如图 3.2 所示。只要出现一些小小的变动，就会使得软件垮掉，而且难于测试。

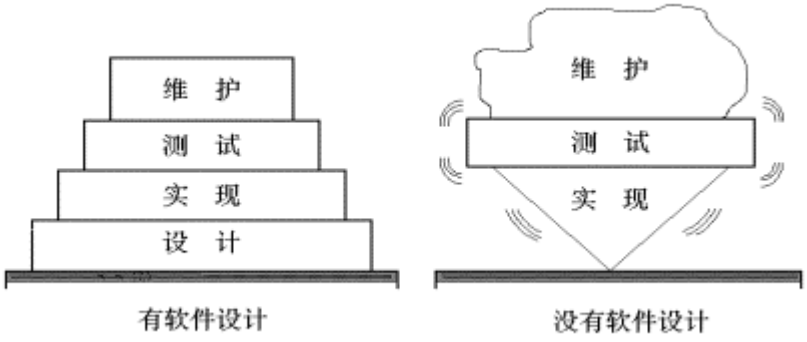


图 4.2 软件设计的重要性

(2) 软件设计的过程

软件设计是一个把软件需求变换成软件表示的过程。最初这种表示只是描绘出可直接反映功能、数据、行为需求的软件的总的框架，然后进一步细化，在此框架中填入细节，把它加工成在程序细节上非常接近于源程序的软件表示。

从工程管理的角度来看，软件设计分两步完成。首先做概要设计，将软件需求转化为数据结构和软件的系统结构，并建立接口。然后是详细设计，即过程设计。通过对结构表示进行细化，得到软件的详细的数据结构和算法。

McGlanghlin 给出在将需求转换为设计时判断设计好坏的三条特征：

§ 设计必须实现分析模型中描述的所有显式需求，必须满足用户希望的所有隐式需求。

§ 设计必须是可读、可理解的，使得将来易于编程、易于测试、易于维护。

§ 设计应从实现角度出发，给出与数据、功能、行为相关的软件全貌。

以上三点就是软件设计过程的目标。为达到这些目标，必须建立衡量设计的技术标准。

① 设计出来的结构应是分层结构，从而建立软件成份之间的控制。

② 设计应当模块化，从逻辑上将软件划分为完成特定功能或子功能的构件。

③ 设计应当既包含数据抽象，也包含过程抽象。

④ 设计应当建立具有具有独立功能特征的模块。

⑤ 设计应当建立能够降低模块与外部环境之间复杂连接的接口。

⑥ 设计应能根据软件需求分析获取的信息，建立可驱动可重复的方法。

软件设计过程根据基本的设计原则，使用系统化的方法和完全的的设计评审来建

立良好的设计。

2. 软件设计的原则

(1) 抽象化

对软件进行模块设计的时候，可以有不同的抽象层次。在最高的抽象层次上，可以使用问题所处环境的语言描述问题的解法。而在较低的抽象层次上，则采用过程化的方法。

§ 过程的抽象 ：在软件工程过程中，从系统定义到实现，每进展一步都可以看做是对软件解决方案的抽象化过程的一次细化。在软件计划阶段，软件被当做整个计算机系统中的一个元素来看待。在软件需求分析阶段，用"问题所处环境的为大家所熟悉的术语"来描述软件的解决方法。而在从概要设计到详细设计的过程中，抽象化的层次逐次降低。当产生源程序时到达最低的抽象层次。

§ 数据抽象 ：数据抽象与过程抽象一样，允许设计人员在不同层次上描述数据对象的细节。例如，可以定义一个 draw 数据对象，并将它规定为一个抽象数据类型，用它的构成元素来定义它的内部细节。此时，数据抽象 draw 本身是由另外一些数据抽象构成。而且在定义 draw 的抽象数据类型之后，就可以引用它来定义其它数据对象，而不必涉及 draw 的内部细节。

§ 控制抽象 ：与过程抽象和数据抽象一样，控制抽象可以包含一个程序控制机制而无须规定其内部细节。控制抽象的例子就是在操作系统中用以协调某些活动的同步信号。

(2) 自顶向下，逐步细化

Niklaus Wirth 提出的设计策略。将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。最初的说明只是概念性地描述了系统的功能或信息，但并未提供有关功能的内部实现机制或有关信息的内部结构的任何信息。设计人员对初始说明仔细推敲，进行功能细化或信息细化，给出实现的细节，划分出若干成份。然后再对这些成份，施行同样的细化工作。随着细化工作的逐步展开，设计人员就能得到越来越多的细节。

(3) 模块化

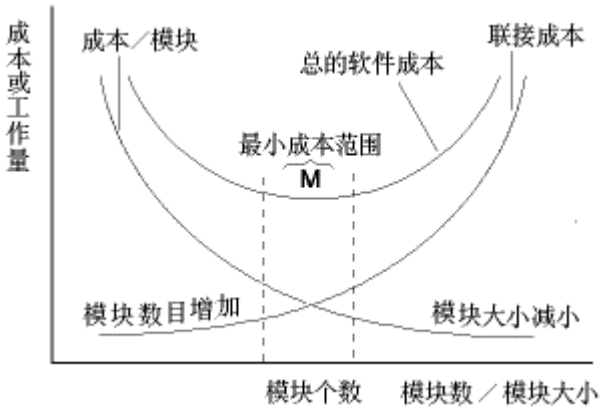
软件系统的层次结构正是模块化的具体体现。就是说，整个软件被划分成若干单独命名和可编址的部分，称之为模块。这些模块可以被组装起来以满足整个问题的需求。

一个大软件，由于其控制路径多、涉及范围广、变量多及其总体复杂性，使其相对于一个较小的软件不容易被人们理解。在解决问题的实践中，如果把两个问题结合起来作为一个问题来处理，其理解复杂性大于这两个问题被分开考虑时的理解复杂性之和。因此，把一个大而复杂的问题分解成一些独立的易于处理的小问题，解决起来就容易得多。

基于上述考虑，把问题 / 子问题（功能 / 子功能）的分解与软件开发中的系统 / 子系统或者系统 / 模块对应起来，就能够把一个大而复杂的软件系统划分成易于理解的比较单纯的模块结构。所谓"比较单纯"，是指模块和其它模块之间的接口应尽可能独立。

实际上，如果模块是相互独立的，当模块变得越小，每个模块花费的工作量越低；但当模块数增加时，模块间的联系也随之增加，把这些模块联接起来的工作量也随之增加。如图 3.3 所示。因此，存在一个模块个数 M， 它使得总的开发成本达到最小。

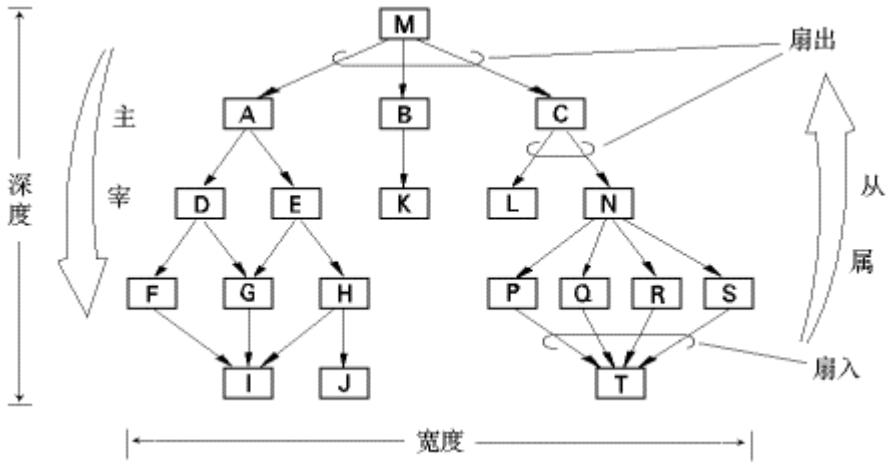
图 3.3 模块大小、模块数目与费用的关系



(4) 控制层次

控制层次也叫做程序结构，它表明了程序构件（模块）的组织情况。控制层次往往用程序的层次（树形或网状）结构来表示。如图 4.4 所示。位于最上层根部是顶层模块，它是程序的主模块。与其联系的有若干下属模块，各下属模块还可以进一步引出更下一层的下属模块。模块 M 是顶层模块，如果算做第 0 层，则其下属模块 A、B 和 C 为第 1 层，模块 D、E、K、L 和 N 是第 2 层，…，等等。

图 3.4 程序的层次结构图示例



§ 程序结构的深度：程序结构的层次数称为结构的深度。结构的深度在一定意义上反映了程序结构的规模和复杂程度。

§ 程序结构的宽度：层次结构中同一层模块的最大模块个数称为结构的宽度。

§ 模块的扇入和扇出：扇出表示一个模块直接调用（或控制）的其它模块数目。扇入则定义为调用（或控制）一个给定模块的模块个数。多扇出意味着需要控制和协调许多下属模块。而多扇入的模块通常是公用模块。

要注意的是，程序结构是软件的过程表示，但并未表明软件的某些过程性特征。比如，进程序列、事件／决策的顺序或其它的软件动态特性。

(5) 结构划分

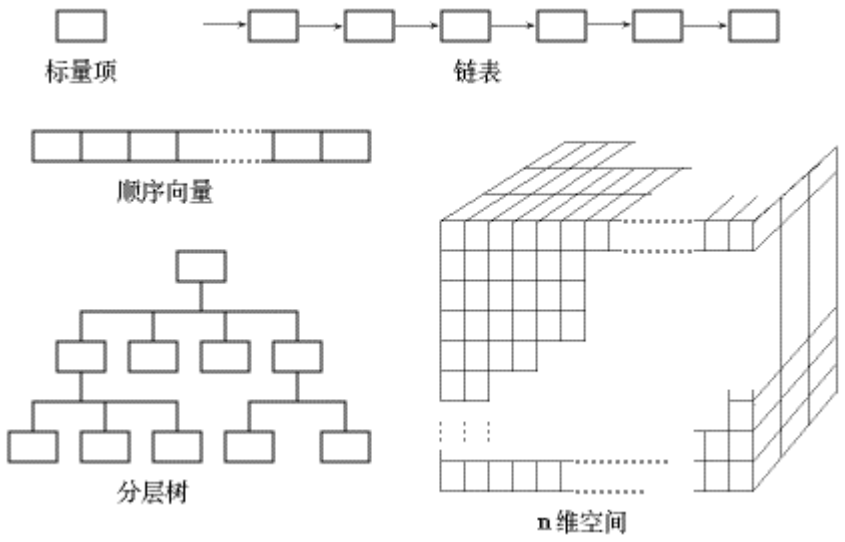
程序结构可以按水平方向或垂直方向进行划分。水平划分按主要的程序功能来定义模块结构的各个分支。顶层模块是控制模块，用来协调程序各个功能之间的通信和运行。其下级模块的最简单的水平划分方法是建立三个分支：输入、处理（数据变换）和输出。这种划分的优点是：由于主要的功能相互分离，易于修改、易于扩充，且没有副作用。缺点是：需要通过模块接口传递更多的数据，使程序流的整体控制复杂化。

垂直划分也叫做因子划分。主要用在程序的体系结构中，且工作自顶向下逐层分布：顶层模块执行控制功能，少做实际处理工作，而低层模块是实际输入、计算和输出的具体执行者。这种划分的优点是：对低层模块的修改不太可能引起副作用的传播，而恰恰对计算机程序的修改常常发生在低层的输入、计算或输出模块中。因此，程序的整体控制结构不太可能被修改，便于将来的维护。

(6) 数据结构

数据结构是数据的各个元素之间的逻辑关系的一种表示。数据结构设计应确定数据的组织、存取方式、相关程度、以及信息的不同处理方法。数据结构的组织方法和复杂程度可以灵活多样，但典型的数据结构种类是有限的，它们是构成一些更复杂结构的基本构件块。图 3.5 表示了这些典型的数据结构。

图 3.5 典型的数据结构



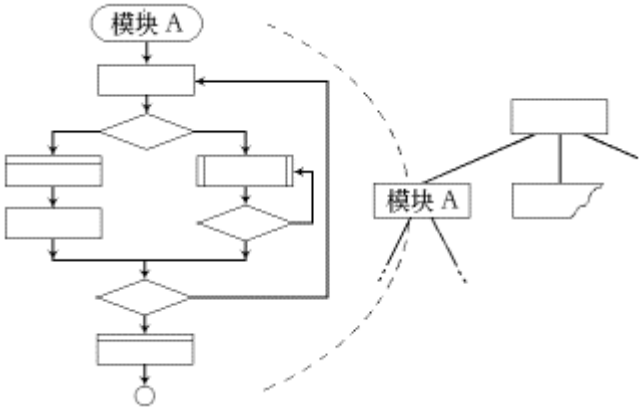
标量是最简单的一种数据结构。所谓标量项就是单个的数据元素，例如一个布尔量、整数、实数或一个字符串。可以通过名字对它们进行存取。

若把多个标量项组织成一个表或者顺序邻接为一组时，就形成了顺序向量。顺序向量又称为一维数组。通常可以通过下标及数组名来访问数组中的某一元素。把顺序向量扩展到二维、三维，直至任意维，就形成了 n 维向量空间。最常见的 n 维向量空间是二维矩阵。

链表是一种更灵活的数据结构，它把不相邻的标量项、向量或空间结构用拉链指针链接起来，使得它们可以像表一样得到处理。

组合上述基本数据结构可以构成其它数据结构。例如，可以用包含标量项、向量或 n 维空间的多重链表来建立分层结构和网络结构。而利用它们又可以实现多种集合的存储。
必须注意，数据结构和程序结构一样，可以在不同的抽象层次上表示。例如，一个栈是一种线性结构的逻辑模型，其特点是只允许在结构的一端进行插入或删除运算。它可以用向量实现，也可以用链表实现。
(7) 软件过程

图 3.6 一个模块内的软件过程



程序结构描述了整个程序的控制层次关系和各个部分的接口情况，而图 3.6 所示的软件过程则着重描述各个模块的处理细节。
软件过程必须提供精确的处理说明，包括事件的顺序、正确的判定点、重复的操作直至数据的组织和结构等等。程序结构与软件过程是有关系的。对每个模块的处理必须指明该模块所在的上下级环境。软件过程遵从程序结构的主从关系，因此它也是层次化的。
(8) 信息隐蔽
如何分解一个软件才能得到最佳的模块组合？为了明确怎样去做，需要了解什么是"信息隐蔽"。由 parnas 提倡的信息隐蔽是指，每个模块的实现细节对于其它模块来说是隐蔽的。就是说，模块中所包含的信息（包括数据和过程）不允许其它不需要这些信息的模块使用。
通常有效的模块化可以通过定义一组独立的模块来实现，这些模块相互间的通信仅使用对于实现软件功能来说是必要的信息。通过抽象，帮助我们确定组成软件的过程（或信息）实体，通过信息隐蔽，则可定义和实施对模块的过程细节和局部数据结构的存取限制。
由于一个软件系统在整个软件生存期内要经过多次修改，所以在划分模块时要采取措施，使得大多数过程和数据对软件的其它部分是隐蔽的。这样，在将来修改软件时偶然引入错误所造成的影响就可以局限在一个或几个模块内部，不致波及到软件的其它部分。
3. 软件体系结构
软件体系结构的三要素是程序构件（模块）的层次结构、构件之间交互的方式，以及数据的结构。软件设计的一个目标是建立软件的体系结构表示。将这个表示当作一个框架，从事更详细的设计活动。Shaw 和 Garlan 提出了在软件体系结构

设计中应保持的几个性质：
§ 结构 ：体系结构设计应当定义系统的构件，以及这些构件打包的方式和相互交互的方式。如将对象打包以封装数据和操纵数据的处理，并通过相关操作的调用来进行交互。
§ 附属的功能 ：体系结构设计应当描述设计出来的体系结构如何实现对功能、性能、可靠性、安全性、适应性，以及其它的系统需求。
§ 可复用 ：体系结构设计应当描述为一种可复用的模式，以便在以后类似的系统族的设计中使用它们。此外，设计应能复用体系结构中的构造块。
表 3.1 列出可能的软件构件，表 3.2 列出可能的构件间的连接方式。

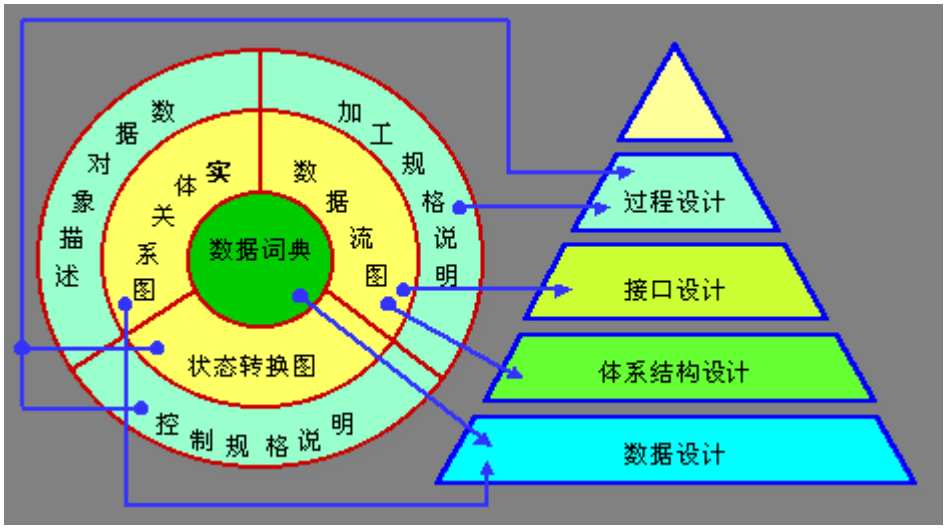


表 3.2 构件之间的连接方式

连接	特点与示例
过程调用	在某一个执行路径中传递执行指针。例如，普通过程调用（同一个命名空间）、远程过程调用（不同的命名空间）。
数据流	相互独立的处理通过数据流进行交互，在得到数据的同时被赋予控制权限。例如，UNIX 系统中的管道（pipes）。
间接激活	处理是因事件的发生而激活的，在处理之间没有直接的交互。例如，事件驱动系统、自动垃圾回收等。
消息传递	相互独立的处理之间有明确的交互，通过显式的离散方式的数据传递。这种传递可以是同步的，也可以是异步的。例如，TCP／IP。
共享数据	构件们通过同一个数据空间进行并发的操作。例如，多用户数据库、数据黑板系统。

软件系统的体系结构经历了一个由低级到高级的发展过程。其间出现过下列体系结构。
(1) 数据流系统
§ 批处理
§ 管道及过滤器
这种结构中的每一个组成成份都有一套输入和输出数据，都依输入数据－处理－输出结果的方式工作。进行数据变换的构件叫做过滤器，把数据从一个过滤器的

输出导入到另一个过滤器的输入，就叫做管道。在这种系统中，各个过滤器必须是相互独立的，每一个过滤器对它的上游或下游的过滤器的情况是不知道的，也不能做任何假设。如果要求最终的输出结果与各个过滤器的执行次序相关，就是一个数据流方式的体系结构。这种结构的优点是：数据流程设计明确，直接支持复用，系统容易维护和升级，可以进行某些性能分析（如流量、死锁等），容易支持并行计算。缺点是：容易导致把系统变成为简单的批处理作业。每一个过滤器都要考虑相似的数据检验和处理，不能很好地支持交互式操作和反馈。对数据的格式不能做过多的约定，使得每个过滤器的实现会更复杂。

(2) 调用－返回系统

§ 主程序 / 子程序

§ 层次结构

在层次结构中，每一层都只与上下相邻的两层通信。每一层在利用下层基础服务的条件下，为上层提供服务。最典型的例子就是各种虚拟机、X-window 以及 OSI-ISO 的 7 层网络协议。这种结构的优点是：提供逐步抽象的编程支持，支持复用及系统升级。缺点是：不是所有的系统都适合于建成层次结构，不能提供最佳性能。

§ 面向对象的系统

在这种系统中，数据和其相关的基本操作被封装在一起。系统的构件是对象。对象具有诸如封装、隐蔽、继承等良好的特性。对象必须自己维护其数据的一致性。这种结构的优点是：将具体的实现部分隐蔽在对象中，使得代码之间的独立性很好，有利于将复杂的系统分解为相互操纵的子任务。缺点是：对象间进行一般的调用时必须知道对方的标识。如果一个对象的标识发生变化，所有显式调用这个对象操作的地方都要修改。对象之间的同步等还缺乏现成的机制。

(3) 独立构件系统

§ 进程间通信

§ 事件驱动

这种结构的特点是事件的发出者不必知道，也不应假设对该事件的具体处理过程。它的优点是：提供了强大的可复用性支持。系统的重配置也很容易。缺点是：软件系统中的构件在很大程度上依靠操作系统的调用。如果一个事件的处理需要多个进程处理，活动进程的激活次序是不能保证的。经过事件传递数据也较受限制。事件处理中的逻辑处理实际上带有时态性质，与一般的逻辑处理不一样。

(4) 虚拟机

§ 解释器

解释器的目的是实现一个虚拟机。这是一类比较复杂的体系结构，但是，如果进行结构分析的话，就会发现看似不同的系统有着十分相似的结构。

§ 推理系统

§ 过程控制

这种结构要比较外界变量与目标常数的差异，经过控制策略的计算，反馈信号，控制需要监测和控制的过程。

(5) 数据为中心的系统

§ 数据库

§ 超文本系统

§ 数据黑板系统

这种结构的特点是：有两种构件：一是被共享的结构化数据，保存了所有的运行

状态；二是所有访问这些数据的独立的进程。如果是因为输入的数据而引起对共享数据的操作，那么这种控制策略下的体系结构就叫做数据库。如果是由共享数据的当前状态触发相应的处理进程，那么这种体系结构就叫做数据黑板。许多表面上看起来是其它种类的体系结构，可以同时归入这种体系结构。

(6) 其它

§ 分布式处理的系统

§ 特定领域的软件体系结构

(7) 演进与综合

一般来讲，一个新系统的原型最开始出现时，往往以批处理方式进行组合；在进一步的应用分析过程中，将逐步对交互控制方式、实时的反馈、集成化等方面提出需求，因而系统将逐步向以数据为中心的系统过渡，中间还可能会经过一种或几种演化形态。

4. 有效的模块设计

模块化方法带来了许多好处。一方面，模块化设计降低了系统的复杂性，使得系统容易修改；另一方面，推动了系统各个部分的并行开发，从而提高了软件的生产效率。

(1) 模块

模块又称构件，在传统的方法中指用一个名字就可调用的一段程序。类似于高级语言中的过程、函数等。它一般具有如下三个基本属性：

§ 功能：即指该模块实现什么功能，做什么事情。

§ 逻辑：即描述模块内部怎么做。

§ 状态：即该模块使用时的环境和条件。

在描述一个模块时，还必须按模块的外部特性与内部特性分别描述。模块的外部特性是指模块的模块名、参数表、以及给程序以至整个系统造成的影响。而模块的内部特性则是指完成其功能的程序代码和仅供该模块内部使用的数据。

对于模块的外部环境（例如需要调用这个模块的上级模块）来说，只需要了解这个模块的外部特性足够了，不必了解它的内部特性。而软件设计阶段，通常是先确定模块的外部特性，然后再确定它的内部特性。

(2) 模块独立性

所谓模块的独立性，是指软件系统中每个模块只涉及软件要求的具体的子功能，而和软件系统中其它的模块的接口是简单的。例如，若一个模块只具有单一的功能且与其它模块没有太多的联系，那么，我们则称此模块具有模块独立性。

一般采用两个准则度量模块独立性。即模块间的耦合和模块的内聚。

(3) 内聚性

内聚是模块功能强度（一个模块内部各个元素彼此结合的紧密程度）的度量。一个内聚程度高的模块（在理想情况下）应当只做一件事。一般模块的内聚性分为七种类型。



在上面的关系中可以看到，位于高端的几种内聚类型最好，位于中段的几种内聚类型是可以接受的，但位于低端的内聚类型很不好，一般不能使用。因此，人们总是希望一个模块的内聚类型向高的方向靠。模块的内聚在系统的模块化设计中是一个关键的因素。

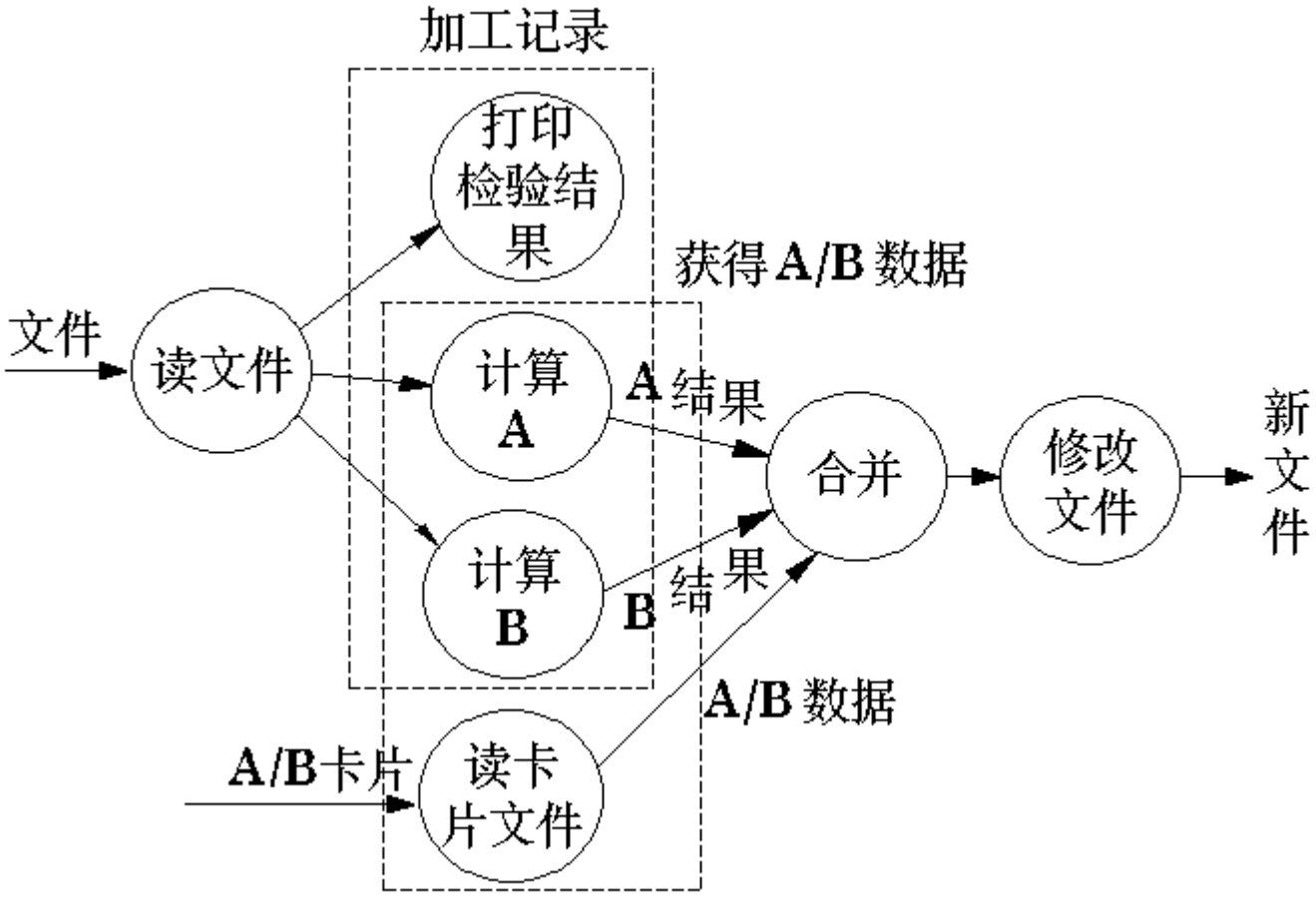
§ 巧合内聚（偶然内聚）：当几个模块内凑巧有一些程序段代码相同，又没有明确表现出独立的功能，把这些代码独立出来建立的模块即为巧合内聚模块。它是内聚程度最低的模块。缺点是模块的内容不易理解，不易修改和维护。

§ 逻辑内聚 ：这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的控制型参数来确定该模块应执行哪一种功能。逻辑内聚模块比巧合内聚模块的内聚程度要高。因为它表明了各部分之间在功能上的相关关系。

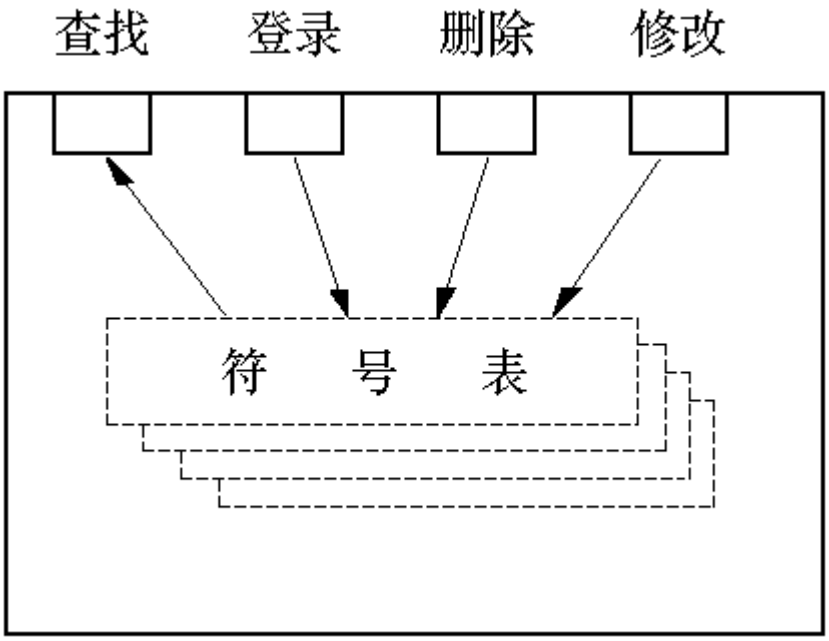
§ 时间内聚（经典内聚）：这种模块大多为多功能模块，但要求模块的各个功能必须在同一时间段内执行。例如初始化模块和终止模块。时间内聚模块比逻辑内聚模块的内聚程度又稍高一些。在一般情形下，各部分可以以任意的顺序执行，所以它的内部逻辑更简单。

§ 过程内聚 ：使用流程图做为工具设计程序的时侯，常常通过流程图来确定模块划分。把流程图中的某一部分划出组成模块，就得到过程内聚模块。这类模块的内聚程度比时间内聚模块的内聚程度更强一些。

§ 通信内聚 ：如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称之为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。如图 3.7 所示。



§ 信息内聚（顺序内聚）：这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。例如，图 3.8 所示的模块具有 4 个功能，由于模块的所有功能都是基于同一个数据结构（符号表），因此，它是一个信息内聚的模块。



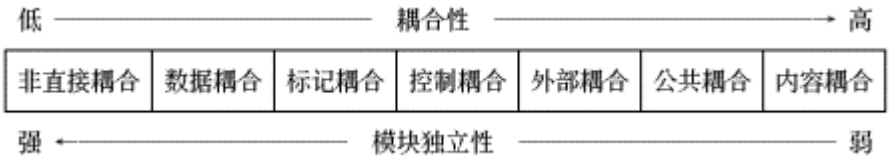
信息内聚模块可以看成是多个功能内聚模块的组合，并且达到信息的隐蔽。即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。当把程序某些方面细节隐藏在一个模块中时，就增加了模块的独立性。

§ 功能内聚 ：一个模块中各个部分都是为完成一项具体功能而协同工作，紧密联系，不可分割的。则称该模块为功能内聚模块。功能内聚模块时内聚性最强的模块。

(4) 耦合性

耦合是模块之间的相对独立性（互相连接的紧密程度）的度量。它取决于各个模块之间接口的复杂程度、调用模块的方式以及哪些信息通过接口。

一般模块之间可能的连接方式有七种，构成耦合性的七种类型。



§ 内容耦合 ：如果一个模块直接访问另一个模块的内部数据；或者一个模块不通过正常入口转到另一模块内部；或者两个模块有一部分程序代码重迭；或者一个模块有多个入口，则两个模块之间就发生了内容耦合。在内容耦合的情形，被访问模块的任何变更，或者用不同的编译器对它再编译，都会造成程序出错。这种耦合是模块独立性最弱的耦合。

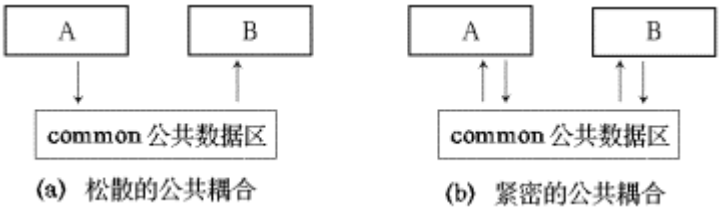
§ 公共耦合 ：若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。

公共耦合的复杂程度随耦合模块的个数增加而显著增加。如图 4.9 所示，若只是两个模块之间有公共数据环境，则公共耦合有两种情况：松散公共耦合和紧密公

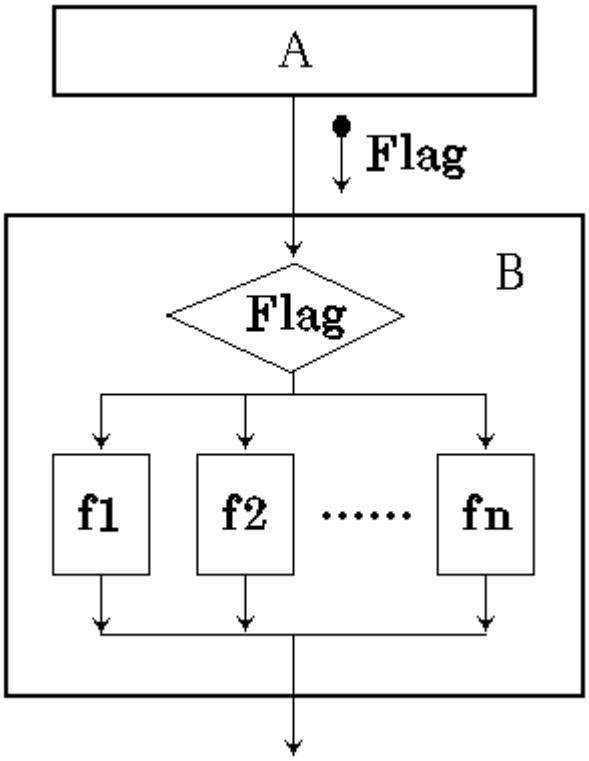
共耦合。只有在模块之间共享的数据很多，且通过参数表传递不方便时，才使用公共耦合。

§ 外部耦合 ：一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。外部耦合引起的问题类似于公共耦合，区别在于在外部耦合中不存在依赖于一个数据结构内部各项的物理安排。

图 3.9 公共耦合



§ 控制耦合 ：如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。如图 3.10 所示。



这种耦合的实质是在单一接口上选择多功能模块中的某项功能。因此，对被控制模块的任何修改，都会影响控制模块。另外，控制耦合也意味着控制模块必须知道被控制模块内部的一些逻辑关系，这些都会降低模块的独立性。

§ 标记耦合 ：如果一组模块通过参数表传递记录信息，就是标记耦合。事实上，这组模块共享了某一数据结构的子结构，而不是简单变量。这要求这些模块都必

须清楚该记录的结构，并按结构要求对记录进行操作。

§ 数据耦合 ：如果一个模块访问另一个模块时，彼此之间是通过数据参数（不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的，则称这种耦合为数据耦合。数据耦合是松散的耦合，模块之间的独立性比较强。

§ 非直接耦合 ：如果两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的，这就是非直接耦合。这种耦合的模块独立性最强。

实际上，开始时两个模块之间的耦合不只是一种类型，而是多种类型的混合。这就要求设计人员进行分析、比较，逐步加以改进，以提高模块的独立性。

模块之间的连接越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。一个模块内部各个元素之间的联系越紧密，则它的内聚性就越高，相对地，它与其它模块之间的耦合性就会减低，而模块独立性就越强。因此，模块独立性比较强的模块应是高内聚低耦合的模块。

5. 结构化设计方法（Structured Design, SD）

从系统设计的角度出发，软件设计方法可以分为三大类。第一类是根据系统的数据流进行设计，称为面向数据流的设计或者过程驱动的设计，以结构化设计方法为代表。第二类是根据系统的数据结构进行设计，称为面向数据结构的设计或者数据驱动的设计，以 LCP（程序逻辑构造）方法、Jackson 系统开发方法和数据结构化系统开发（DSSD）方法为代表。第三类设计方法即面向对象的设计。

结构化设计方法是基于模块化、自顶向下细化、结构化程序设计等程序设计技术基础上发展起来的。该方法实施的要点是：① 建立数据流的类型。② 指明流的边界。③ 将数据流图映射到程序结构。④ 用"因子化"方法定义控制的层次结构。⑤ 用设计测量和一些启发式规则对结构进行细化。

(1) 在系统结构图（SC）中的模块

在系统结构图中不能再分解的底层模块为原子模块。如果一个软件系统的全部实际加工（数据计算或处理）都由底层的原子模块来完成，而其它所有非原子模块仅仅执行控制或协调功能，这样的系统就是完全因子分解的系统。如果系统结构图是完全因子分解的，就是最好的系统。一般地，在系统结构图中有 4 种类型的模块：

§ 传入模块 ：从下属模块取得数据，经过某些处理，再将其传送给上级模块。

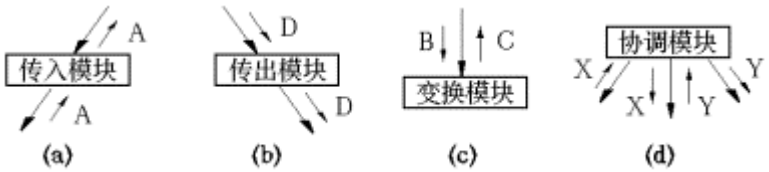
§ 传出模块 ：从上级模块获得数据，进行某些处理，再将其传送给下属模块。

§ 变换模块 ：即加工模块。它从上级模块取得数据，进行特定的处理，转换成其它形式，再传送回上级模块。大多数计算模块（原子模块）属于这一类。

§ 协调模块 ：对所有下属模块进行协调和管理的模块。 在系统的输入 / 输出部分或数据加工部分可以找到这样的模块。在一个好的系统结构图中，协调模块应在较高层出现。

在实际系统中，有些模块属于上述某一类型，还有一些模块是上述各种类型的组合。

图 3.11 系统结构图的四种模块类型



(2) 变换流与变换型系统结构

变换型数据处理问题的工作过程大致分为三步，即取得数据，变换数据和给出数据。如图 3.12 所示。这三步反映了变换型问题数据流的基本思想。其中，变换数据是数据处理过程的核心工作，而取得数据只不过是为它做准备，给出数据则是对变换后的数据进行后处理工作。

图 3.12 变换型数据流



变换型系统结构图如图 3.13 所示，相应于取得数据、变换数据、给出数据，系统的结构图由输入、中心变换和输出等三部分组成。

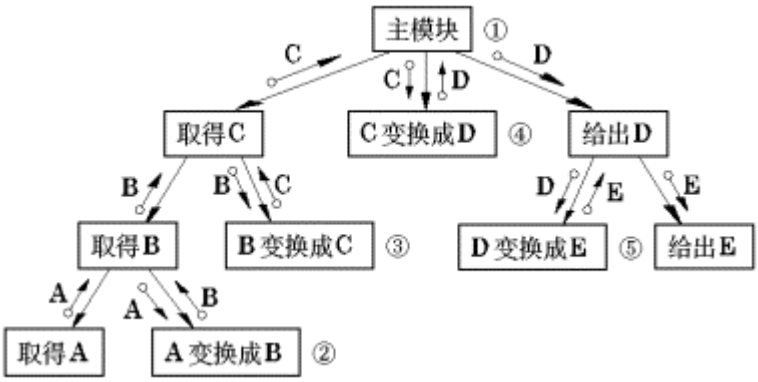
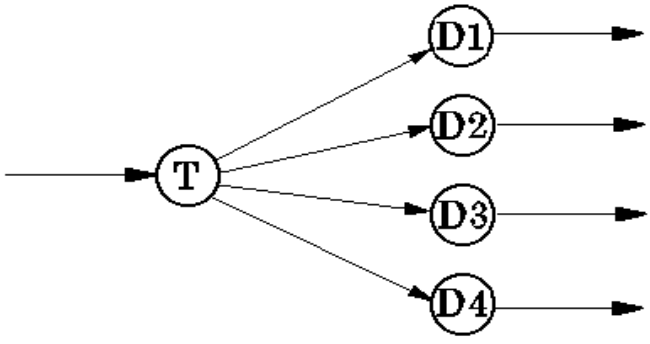


图 3.13 变换型的系统结构图

(3) 事务流与事务型系统结构图

事务型数据处理问题的工作机理是接受一项事务，根据事务处理的特点和性质，选择分派一个适当的处理单元，然后给出结果。我们把完成选择分派任务的部分叫做事务处理中心。或分派部件。这种事务型数据处理问题的数据流图如图 3.14 所示。



其中，输入数据流在事务中心 T 处做出选择，激活某一种事务处理加工。D1—D4 是并列的供选择的事务处理加工。

事务型数据流图所对应的系统结构图就是事务型系统结构图。如图 3.15 所示。

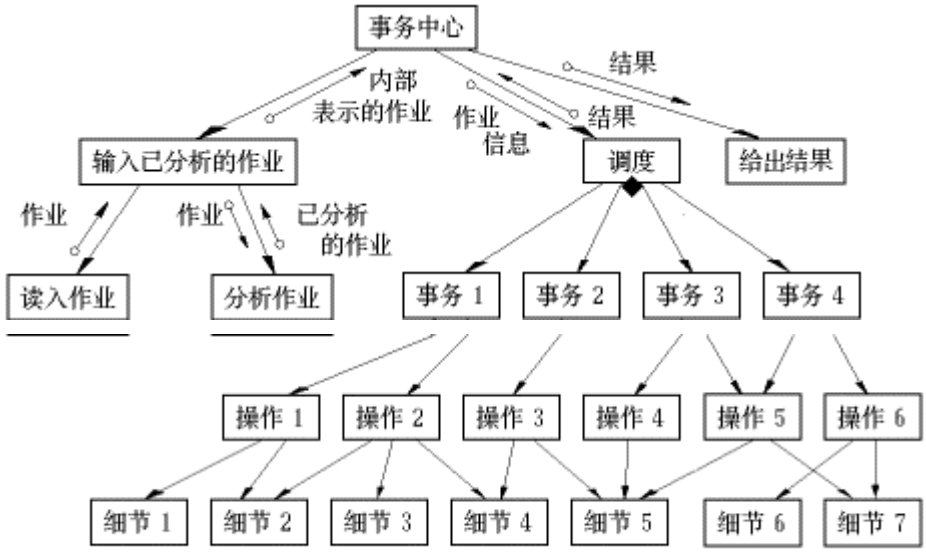


图 3.15 事务型系统结构图

在事务型系统结构图中，事务中心模块按所接受的事务的类型，选择某一个事务处理模块执行。各个事务处理模块是并列的，依赖于一定的选择条件，分别完成不同的事务处理工作。每个事务处理模块可能要调用若干个操作模块，而操作模块又可能调用若干个细节模块。不同的事务处理模块可以共享一些操作模块。同样，不同的操作模块又可以共享一些细节模块。

事务型系统结构图在数据处理中经常遇到，但是更多的是变换型与事务型系统结构图的结合。例如，变换型系统结构中的某个变换模块本身又具有事务型的特点。

(4) 变换映射

变换映射是体系结构设计的一种策略。运用变换映射方法建立初始的变换型系统结构图，然后对它做进一步的改进，最后得到系统的最终结构图。设计的步骤如下。

步骤 1：复审基本系统模型（0 层数据流图和支持信息）。评估系统规格说明和软件需求规格说明。

步骤 2：复审和细化软件的数据流图。重画数据流图时， 可以从物理输入到物理输出，或者相反. 还可以从顶层加工框开始，逐层向下。

步骤 3：确定数据流图中含有变换流特征还是含有事务流特征。通常，系统的信息流总能表示为变换型，但其中也可能遇到明显的事务流特征，这时可采用变换型为主，在局部范围采用事务型的设计方法。

步骤 4：区分输入流、输出流和中心变换部分，即标明流的边界。不同的设计人员可能选择不同的流边界，这将导致不同的系统结构图。

步骤 5：进行一级"因子化"分解，设计顶层和第一层模块。

首先设计主模块，用程序名字为它命名，将它画在与中心变换相对应的位置上。做为系统的顶层，它调用下层模块，完成系统所要做的各项工作。系统结构第一层的设计方针：为每一个逻辑输入设计一个输入模块，它为主模块提供数据；为

每一个逻辑输出设计一个输出模块，它将主模块提供的数据输出；为中心变换设计一个变换模块，它将逻辑输入转换成逻辑输出。第一层模块与主模块之间传送的数据应与数据流图相对应。

步骤 6：进行二级"因子化"分解，设计中、下层模块。这一步工作是自顶向下，逐层细化，为每一个输入模块、输出模块、变换模块设计它们的从属模块。

输入模块要向调用它的上级模块提供数据，因而它必须有两个下属模块：一个是接收数据；另一个是把这些数据变换成它的上级模块所需的数据。输出模块是从调用它的上级模块接收数据，用以输出，因而也应当有两个下属模块：一个是将上级模块提供的数据变换成输出的形式；另一个是将它们输出。中心变换模块的下层模块没有通用的设计方法，一般应参照数据流图的中心变换部分和功能分解的原则来考虑如何对中心变换模块进行分解。

步骤 7：利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。这些启发式原则大致有：

① 模块功能的完善化。一个完整的功能模块，不仅应能完成指定的功能，而且还应当能够告诉使用者完成任务的状态，以及不能完成的原因。

② 消除重复功能，改善软件结构。在系统的初始结构图得出之后，应当审查分析这个结构图。如果发现几个模块的功能有相似之处，可以加以改进。

③ 模块的作用范围应在控制范围之内。模块的控制范围包括它本身及其所有的从属模块。模块的作用范围是指模块内一个判定的作用范围，凡是受这个判定影响的所有模块都属于这个判定的作用范围。如果一个判定的作用范围包含在这个判定所在模块的控制范围之内，则这种结构是简单的，否则，它的结构是不简单的。

④ 尽可能减少高扇出结构，经验证明，一个设计得很好的软件模块结构，通常上层扇出比较高，中层扇出较少，底层扇入到有高扇入的公用模块中。

⑤ 避免或减少使用病态联接。应限制使用如下三种病态联接：直接病态联接（内容耦合）、公共数据域病态联接（公共耦合）和通过通信模块联接。

⑥ 模块的大小要适中。限制模块的大小是减少复杂性的手段之一，因而要求把模块的大小限制在一定的范围之内。通常规定其语句行数在 50～100 左右，最多不超过 500 行。

⑦ 设计功能可预测的模块，但要避免过分受限制的模块。

一个功能可预测的模块不论内部处理细节如何，但对相同的输入数据，总能产生同样的结果。但是，如果模块内部蕴藏有一些特殊的鲜为人知的功能时，这个模块就可能是不可预测的。对于这种模块，如果调用者不小心使用，其结果将不可预测。调用者无法控制这个模块的执行，或者不能预知将会引起什么后果，最终会造成混乱。

为了能够适应将来的变更，软件模块中局部数据结构的大小应当是可控制的，调用者可以通过模块接口上的参数表或一些预定义外部参数来规定或改变局部数据结构的大小。另外，控制流的选择对于调用者来说，应当是可预测的。而与外界的接口应当是灵活的，也可以用改变某些参数的值来调整接口的信息，以适应未来的变更。

⑧ 软件包应满足设计约束和可移植性。

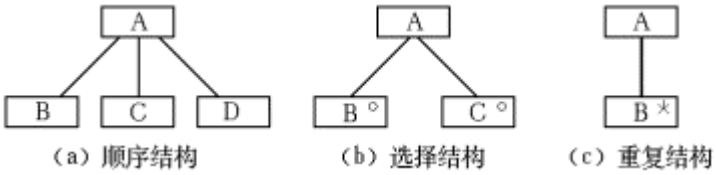
(5) 事务映射

在很多应用中，存在某种作业数据流，它可以引发一个或多个处理。这种数据流就叫做事务。与变换映射类似，事务映射也是从分析数据流图开始，自顶向下，

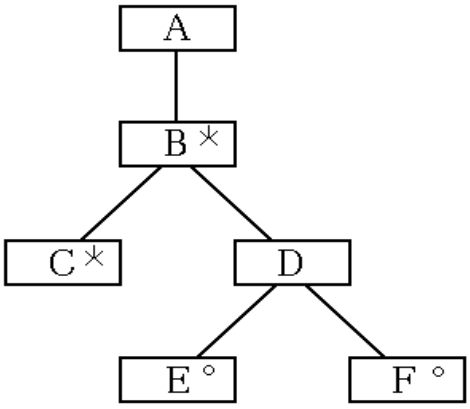
逐步分解，建立系统结构图。所不同的是由数据流图映射成的系统结构图不同。
步骤 1：复审基本系统模型。
步骤 2：复审和细化软件的数据流图。
步骤 3：确定数据流图中含有变换流特征还是含有事务流特征。以上三步与变换映射中的相应工作相同。
步骤 4：识别事务中心和每一条操作路径上的流特征。事务中心通常位于几条操作路径的起始点上，可以从数据流图上直接找出来。输入路径必须与其它所有操作路径区分开来。
步骤 5：将数据流图映射到事务型系统结构图上。事务流应映射到包含一个输入分支和一个分类事务处理分支的程序结构上。输入分支结构的开发与变换流的方法类似。分类事务处理分支结构包含一个调度模块，它调度和控制下属的操作模块。
步骤 6："因子化"分解和细化该事务结构和每一条操作路径的结构。每一条操作路径的数据流图由它自己的信息流特征，可以是变换流也可以是事务流。与每一条操作路径相关的子结构可以依照前面介绍的设计步骤进行开发。
步骤 7：利用一些启发式原则来改进系统的初始结构图。
(6) 注意"黑箱"技术的使用。
在设计当前模块时， 先把这个模块的所有下层模块定义成"黑箱"，并在系统设计中利用它们，暂时不考虑它们的内部结构和实现方法。在这一步定义好的"黑箱"，由于已确定了它的功能和输入、输出，在下一步就可以对它们进行设计和加工。这样，又会导致更多的"黑箱"。最后，全部"黑箱"的内容和结构应完全被确定。这就是我们所说的自顶向下，逐步求精的过程。使用黑箱技术的主要好处是使设计人员可以只关心当前的有关问题，暂时不必考虑进一步的琐碎的次要的细节，待进一步分解时才去关心它们的内部细节与结构。
6. Jackson 系统开发方法
Jackson 方法是一种典型的面向数据结构的分析与设计方法。早期的 Jackson 方法用于小系统的设计，称之为 Jackson 结构程序设计方法，简称 JSP 方法。它是按输入、输出和内部信息的数据结构进行软件设计的，即把数据结构的描述映射成程序结构描述。若数据结构有重复性，则对应程序一定有循环控制结构；若数据结构具有选择性，则对应程序一定需要有判定控制结构，以此揭示数据结构和程序结构之间的内在关系，设计出反映数据结构的程序结构。JSP 方法的的三步曲是信息 → 数据结构 → 程序结构，这三步曲减少了设计决策上的盲目性。但是，当把 JSP 方法用于大系统设计时，就会出现大量复杂的难以对付的结构冲突。因此，促使 M.J.Jackson 提出了 JSD 方法，即 Jackson 系统开发方法。
(1) JSD 方法的步骤
Jackson 系统开发方法把分析的重点放在构造与系统相关联的现实世界，并建立现实世界的信息域的模型上。它实际上是支持软件分析与设计的一组连续的技术步骤。而且，JSD 方法的最终目标是生成软件的过程性描述，没有特别考虑程序模块化结构，模块只是作为过程的副产品而出现，没有特别强调模块独立性。使用 JSD 方法的步骤如下:
① 实体动作分析:从问题的简单描述中，选出软件系统要产生和运用的实体(人、物或组织)，以及现实世界作用于实体上的动作(事件)。实体从名词中选出，动作从动词中选出。当然，只有与问题求解直接有关的实体和动作才能被选出做进一步的分析。

② 实体结构分析：把作用于实体的动作或由实体执行的动作，按时间发生的先后次序排序，并用一个层状的 Jackson 结构图表示。
③ 定义初始模型：把实体和动作表示成一个过程性的模型，定义模型与现实世界的联系。模型系统的规格说明可用系统规格说明图 SSD 来表示。
④ 功能描述：详细说明与已定义的动作相对应的功能。
⑤ 决定系统时间特性：对进程调度特性进行评价和说明。
⑥ 实现：设计组成系统的硬件和软件。
JSD 方法的前三步属于需求分析阶段，后三步属于软件设计阶段。
(2) Jackson 程序设计方法 JSP
JSP 的本质就是"问题应当被分解为可以用三种结构形式表示的构件的层次结构。" Jackson 所说的"结构形式"就是指顺序、选择和重复，实际上，它们就是过程性构造，并成为结构化程序设计方法基础。
① 数据结构表示法
Jackson 提出的数据结构表示有三种基本的构造类型，如图 4.16 所示。图(a)是顺序结构，即数据结构 A 由 B、C、D 三个成分组成且按 B、C、D 顺序排列。图(b)是选择结构，即数据结构 A 或者由 B 组成，或者由 C 组成，二者必具其一。可选择的数据(子结构或数据项)加"° "表示。在图(c)中表示的是重复结构，即数据结构 A 由多个 B 子结构组成，子结构用"*"加以标记。

图 3.16 Jackson 数据结构图



三种基本结构可以组合，形成更复杂的结构体系。如图 3.17 所示。



这种数据结构图可以同样方便地应用于输入、输出和数据库结构。
下面用一个信用卡记账的例子具体说明。信用卡记账系统的输入数据结构是两个实际的账册，它们对应的两个输入文件如图 3.18 所示。

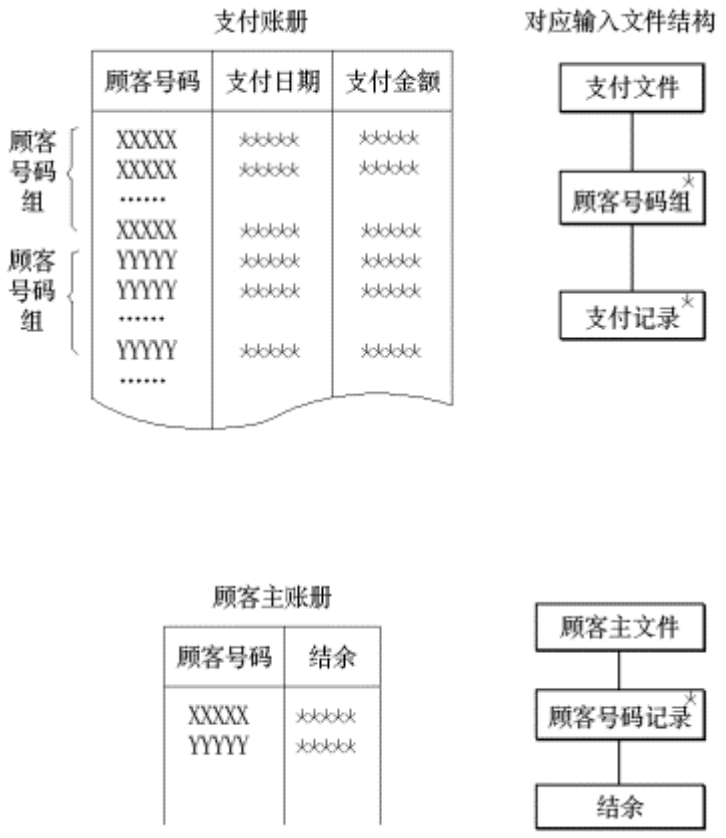


图 3.18 信用卡记账系统的输入

两个实际输入账册都按顾客号码进行登录，所以两个输入文件也是以顾客号码组织记录的。在支付账册中，每个顾客号码行上要登记支付金额、支付日期。则在支付文件中的每个记录中也对应有支付金额(AMT)和支付日期(DATE)项，另外还有标识这个记录的顾客号码(CNO)项。在支付账册中是以顾客号码进行排序的，顾客号码相同的支付记录在支付账册中排列在一起，构成关于该顾客的顾客号码组，在支付文件中也有与之对应的顾客号码组。在顾客主账册中，每一个顾客号码行上登记了顾客号码(CNO)和结余(BAL)，给出某一位顾客的支付能力情况。与之对应的顾客主文件每个记录也有这两项。两个输入文件的内容是一致的。对应这两个输入文件的输入数据结构画在文件图示的左侧。

图 3.19(a) 给出了信用卡记账系统的输出记账报告，在报告中隐含了一个"店方总计"层次。报告中其它部分是顾客信息，因此，隐含了一个与"店方总计"同一层次的"顾客数据"子结构。图 3.19(b) 给出了根据输入和输出数据结构的对应关系建立的输出文件。

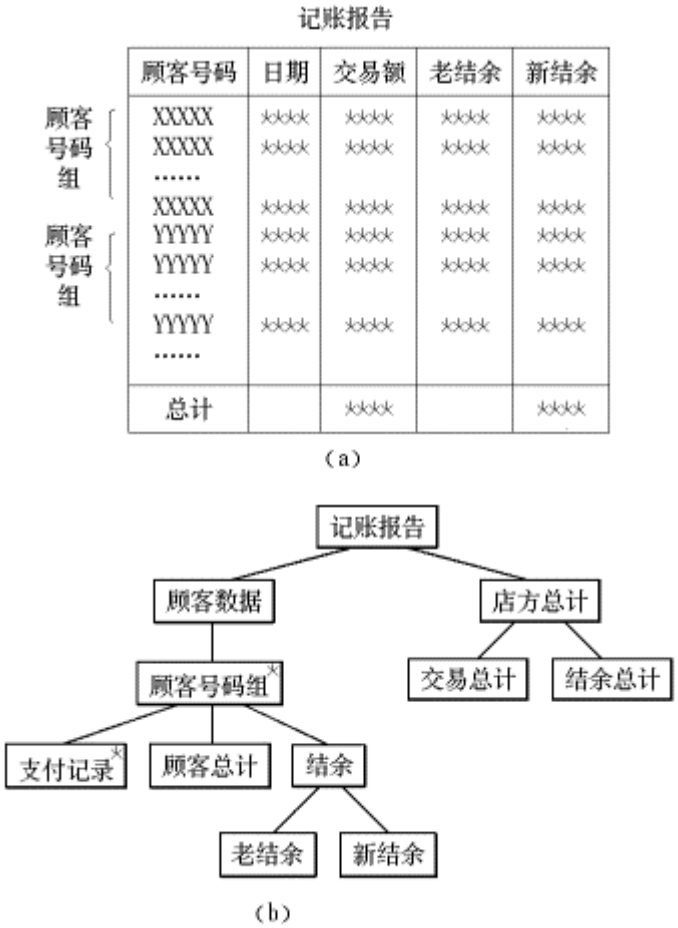


图 3.19 信用卡记账系统的输出

从层次性的输入和(或)输出数据结构可以直接推导出程序或进程的过程性表示。例如从图 3.19(b) 所示的输出文件的 Jackson 数据结构图所导出的程序结构如图 3.20 所示。

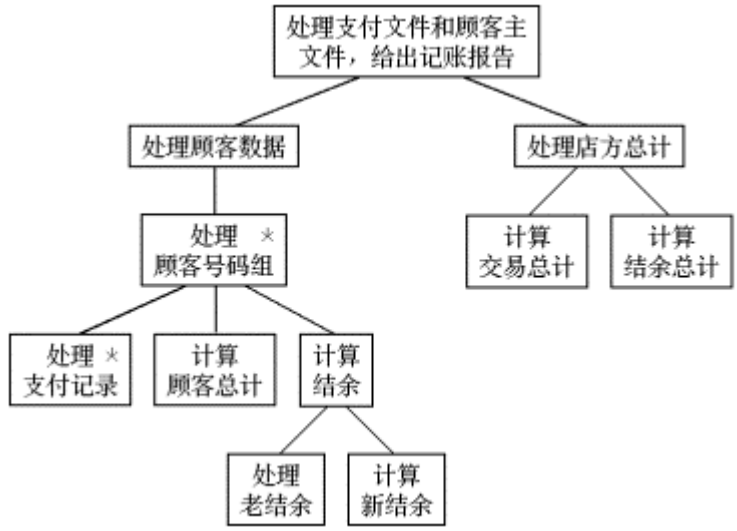


图 3.20 从输出数据结构导出的程序结构

最后，利用 Jackson 给出的三种图解来表示程序或进程的执行逻辑。这种图解类似于程序设计语言，实际上它是一种伪码表示。三种基本控制结构的图解如图 3.21 所示。

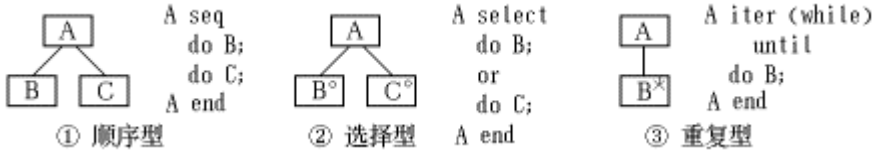


图 3.21 三种基本控制结构的图解

在给出程序的过程性描述时，还需要添加一些必要的可执行操作。例如，打开文件、读文件结束符、读表头数据项、读行数据项、打印符号行、打印数字行、关闭文件等等。把它们分配到程序或进程结构的适当位置，以得到一个完整的过程性描述。对于上例中"处理顾客数据"部分，给出过程性描述如下：

```
PROCESS_CUST_DATA seq
open PAY_FILE;  open CUST_M_FILE;  {分别打开支付文件和顾客主文件}
PROCESS_CNO_GROUP iter until eof: PAY_FILE;  {处理顾客号码组}
read PAY_FILE;  {读支付文件一个记录}
PROCESS_CNO;  {读顾客主文件一个记录，找老结余}
PROCESS_PAY_RECORD iter until end: CNO_GROUP;
{处理顾客号码组中每个支付记录}
write report line;  {写出报告行}
compute total payments;  {计算总支付额}
read PAY_FILE;  {读支付文件下一个记录}
PROCESS_PAY_RECORD end;  {一位顾客数据处理完}
```

```
COMPUTE_CUST_TOTAL;  {计算顾客总数}
COMPUTE_BALANCE seq {计算结余}
PROCESS_OLD_BALANCE;  {处理老结余}
COMPUTE_NEW_BALANCE;  {计算新结余)
write report line;  {写出报告行}
COMPUTE_BALANCE end;  {计算结余完毕}
PROCESS_CNO_GROUP end;  {支付文件处理完成}
PROCESS_CUST_DATA end ;  {Substructure PROCESS_CUST_DATA}
```

用 JSP 方法得到的程序或进程结构图，一般都需要求精和优化。因为这种方法是从输入输出数据结构导出程序结构图，因此有些中间处理过程在结构图中反映不出来。在求精过程中，可以对结构图进行改进和细化，使之完整和易于实现。

(3) 结构冲突

Jackson 把结构冲突定义为三种类型。

① 顺序冲突

顺序冲突是指在输入数据结构和输出数据结构中，输入数据与输出数据的顺序冲突。

例如，仓库存放多种零件 P1，P2，…，每个零件的每次变动(收或发)都有一张卡片做记录。库存管理系统的输入数据结构是一叠卡片组成的文件。文件包括许多零件组，每个零件组又包括许多卡片(变动记录)，每张卡片又分别可以是"收"或"发"，图 3.22(b)。输出数据结构是月报表，见图 3.22(a)。表中列出每种零件的净变化，一种零件的净变化占一行，见图 3.22(c)。月报表来自输入文件，所以它们之间有很好的对应性。

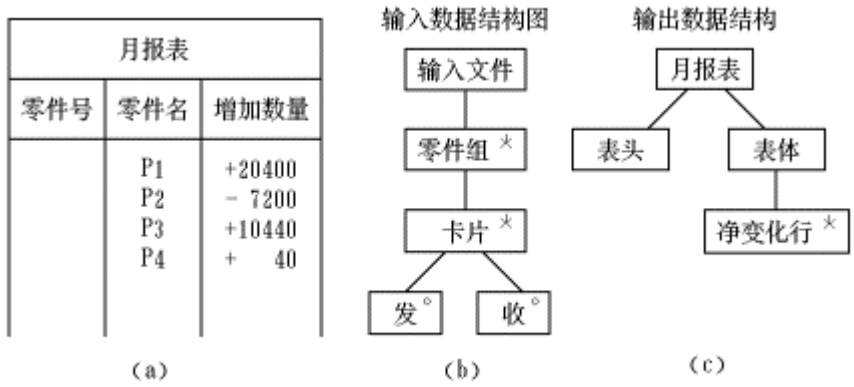


图 3.22 顺序冲突的例子

月报表每一行的内容来自输入文件的每一个零件组，行数与零件组个数相同，排列顺序一致(均按零件号递增次序排列)。这样输入数据与输出数据在内容、数量、顺序上的对应性找到了，也就等于理解了用户所需加工要求，因而很容易导出对应的程序结构。

如果卡片不是按零件组分组，而是按"发"或"收"的日期排列，这样输入数据结构与输出数据结构就找不到上述的对应关系。这种情形就是"顺序冲突"。

② 边界冲突

在输入与输出数据结构中，虽然输入数据与输出数据的顺序相同，但分解不一样。

③ 多重穿插冲突

在输入与输出数据结构中，输入数据与输出数据的顺序相同，而且处于同一程序段中，就象绳索穿绕一样。这些数据并行运行，同时在输入文件中重叠在一起。为了解决上述的结构冲突，Jackson 提出了定义中间数据结构来解决冲突的方法。它可归纳成以下 4 点：

§ 利用数据结构图来定义输入数据结构的特性；
§ 将输入数据结构的元素分解，构造成中间数据结构；
§ 描述输出数据结构的特性；
§ 根据中间数据结构，构造输出数据结构.

这就需要两套程序结构，一个是把输入数据结构转换成中间数据结构；另一个是把中间数据结构转换成输出数据结构。Myers 提出了一种叫做"程序变换"或"多道穿插"的方法，用以解决冲突。 即设计两个子程序，它们既能单独执行，又能并发执行。这样，中间数据结构也可以不要，而使用一个单记录缓冲区来代替它。第一个子程序读入输入记录，把它写到缓冲区中，然后开始等待，直到这个缓冲区变空，然后再把读入的下一个输入记录写到缓冲区中。第二个子程序在等待，待到一个记录被放入缓冲区中之后，它开始读它并进行处理，之后它再等待，直到下一个记录再放入缓冲区后，再读它并进行处理。

例如，用卡片按行输入一个矩阵文件，要求按列打印该矩阵，图 3.23(a)表明这个例子的输入、输出数据结构。从图中可见，卡片文件和打印文件之间存在着对应关系，但行和列的内容不符，行选择和列选择的次序不同。因此，没有办法从这个结构图上构造出相应的程序结构。这就是所谓的结构冲突。

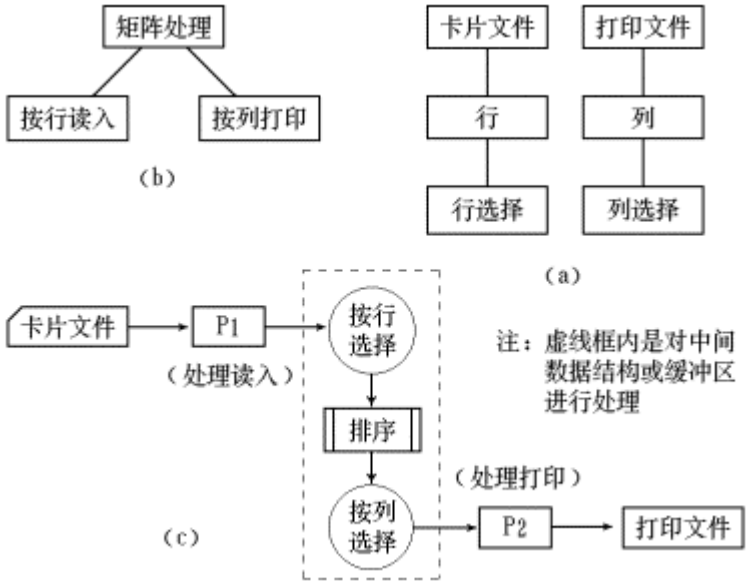


图 3.23 解决冲突的例子

对于这个问题，只能把输入和输出的处理分开来解决。办法有两种。其一是先读入全部卡片，然后再输出，其程序结构图如图 3.23(b)所示。 其二是更一般的方法：构造两个程序结构，分别处理读入和打印，中间有一个排序，如图 3.23(c)所示。

7. 数据设计和文件设计的原则

(1) 数据设计的原则
R.S.Pressman 把数据设计的过程概括成以下两步：
① 为在需求分析阶段所确定的数据对象选择逻辑表示，需要对不同结构进行算法分析，以便选择一个最有效的设计方案。
② 确定对逻辑数据结构所必需的那些操作的程序模块(软件包)，以便限制或确定各个数据设计决策的影响范围。
无论采取什么样的设计方法，如果数据设计得好，往往能产生很好的软件系统结构，具有很强的模块独立性和较低的程序复杂性。
Pressman 提出了一组原则，用来定义和设计数据。
§ 用于软件的系统化方法也适用于数据。应当考虑几种不同的数据组织方案，还应当分析数据设计给软件设计带来的影响。
§ 要确定所有的数据结构和在每种数据结构上施加的操作。对于涉及到软件中若干个功能的实现处理的复杂数据结构，可以为它定义一个抽象数据类型。
§ 应当建立一个数据词典并用它来定义数据和软件的设计。
§ 低层数据设计的决策应推迟到设计过程的后期进行。可以将逐步细化的方法用于数据设计。在需求分析时确定总体数据组织，在概要设计阶段加以细化，而在详细设计阶段才规定具体的细节。
§ 数据结构的表示只限于那些必须直接使用该数据结构内数据的模块才能知道。此原则就是信息隐蔽和与此相关的耦合性原则，把数据对象的逻辑形式与物理形式分开。
§ 数据结构应当设计成为可复用的。建立一个存有各种可复用的数据结构模型的构件库，以减少数据定义和设计的工作量。
§ 软件设计和程序设计语言应当支持抽象数据类型的定义和实现。如果没有直接定义某种复杂数据结构的手段，这种结构的设计和实现往往是很困难的。
以上原则可适用于软件工程的定义阶段和开发阶段。"清晰的信息定义是软件开发成功的关键"。
(2) 文件设计的过程
文件设计指数据存储文件设计，其主要工作就是根据使用要求、处理方式、存储的信息量、数据的活动性，以及所能提供的设备条件等，来确定文件类别，选择文件媒体，决定文件组织方法，设计文件记录格式，并估算文件的容量。
文件设计的过程主要分为两个阶段。第一个阶段是文件的逻辑设计，主要在概要设计阶段实施。它包括：
① 整理必须的数据元素。在软件设计中所使用的数据，有长期的，有短期的，还有临时的。它们都可以存放在文件中，在需要时对它们进行访问。因此首先必须整理应存储的数据元素，给它们一个易于理解的名字，指明其类型和位数，以及其内容涵义。
② 分析数据间的关系。分析在业务处理中哪些数据元素是同时使用的。把同时使用次数多的数据元素归纳成一个文件进行管理。分析数据元素的内容，研究数据元素与数据元素之间的逻辑关系，根据分析，弄清数据元素的含义及其属性。
③ 确定文件的逻辑设计。根据数据关联性分析，明确哪些数据元素应当归于一组进行管理，把应当归于一组的数据元素进行统一布局，产生文件的逻辑设计。
第二个阶段是文件的物理设计，主要在软件的详细设计阶段实施。主要工作有：
④ 理解文件的特性。针对文件的逻辑规格说明，进一步研究从业务处理的观点来看所要求的一些特性，包括文件的使用率、追加率和删除率，以及保护和保密

等。
⑤ 确定文件的存储媒体。选择文件的存储媒体，应当考虑以下一些因素。
§ 数据量：根据处理数据量，估算需要媒体的数量。数据量大的文件可选用磁带、磁盘或光盘做为存储媒体，数据量小的文件可采用软盘做为存储媒体。
§ 处理方式：处理方式有联机处理和批处理。对于联机处理， 多选用直接存取设备，如磁盘等；对于批处理，选用任何一种存储媒体都可以。
§ 存取时间和处理时间：批处理对于时间没有严格的要求， 因此对存储媒体也没有特殊的要求。实时处理最好选用直接存取媒体，如磁盘等，以满足响应时间的要求。
§ 数据结构：根据文件的数据结构，选用能实现其结构的合适媒体及相应的存取方法。例如，顺序文件可选用磁带或光盘，而索引文件和散列文件则必须选用磁盘。
§ 操作要求：对于数据量大，执行时较少要求用户干预的文件，应当选用磁带媒体；而对于频繁交互的文件，应当选用磁盘媒体。
§ 费用要求：在满足上述要求的基础上，应当尽量选用价格低的媒体。
⑥ 确定文件的组织方式。根据文件的特性，来确定文件的组织方式。常用的文件组织方式有：顺序文件（按记录的加入先后次序排列、按记录关键码的升序或降序排列、按记录的使用频率排列）；直接存取文件（无关键码直接存取文件、带关键码直接存取文件、桶式直接存取文件）；索引顺序文件（B＋树）；分区文件；虚拟存储文件；倒排文件等。
⑦ 确定文件的记录格式。确定了文件的组织方式之后，需要进一步确定文件记录中各数据项以及它们在记录中的物理安排。考虑设计记录的布局时，应当注意以下几点：
§ 记录的长度：设计记录的长度要确保能满足需要，还要考虑使用设备的制约和效率，尽可能与读写单位匹配，并尽可能减少处理过程中内外存的交换次数。
§ 数据项的顺序：对于可变长记录，应在记录的开头记入长度信息；对于关键码，应尽量按级别高低，顺序配置；联系较密切的数据项，应归纳在一起进行配置。
§ 数据项的属性：属性相同的数据项，应尽量归纳在一起配置；数据项应按双字长，全字长，半字长和字节的属性，顺序配置。
§ 预留空间：考虑到将来可能的变更或扩充，应当预先留下一些空闲空间。
§ 子数据项：可把一个数据项分成几个子数据项，每一个子数据项也可以做为单独的项来使用。
⑧ 估算存取时间和存储容量（不要求）
8. 过程设计
从软件开发的工程化观点来看，在使用程序设计语言编制程序以前，需要对所采用算法的逻辑关系进行分析，设计出全部必要的过程细节，并给予清晰的表达，使之成为编码的依据。这就是过程设计的任务。
过程设计也叫做详细设计或程序设计，它不同于编码或编程。在过程设计阶段，要决定各个模块的实现算法，并精确地表达这些算法。前者涉及所开发项目的具体要求和对每个模块规定的功能。以及算法的设计和评价，后者需要给出实钡乃惴　枋觯　　擞 μ 峁 乚　躺杓频谋浆锎ぞ摺?br> 表达过程规格说明的工具叫做详细设计工具，它可以分为三类：图形工具、表格工具和语言工具。
(1) 程序流程图

程序流程图独立于任何一种程序设计语言，比较直观、清晰，易于学习掌握。但流程图也存在一些严重的缺点。例如流程图所使用的符号不够规范，常常使用一些习惯性用法。特别是表示程序控制流程的箭头可以不受任何约束，随意转移控制。这些现象显然是与软件工程化的要求相背离的。为了消除这些缺点，应对流程图所使用的符号做出严格的定义，不允许人们随心所欲地画出各种不规范的流程图。例如，为使用流程图描述结构化程序，必须限制流程图只能使用图 3.24 所给出的五种基本控制结构。

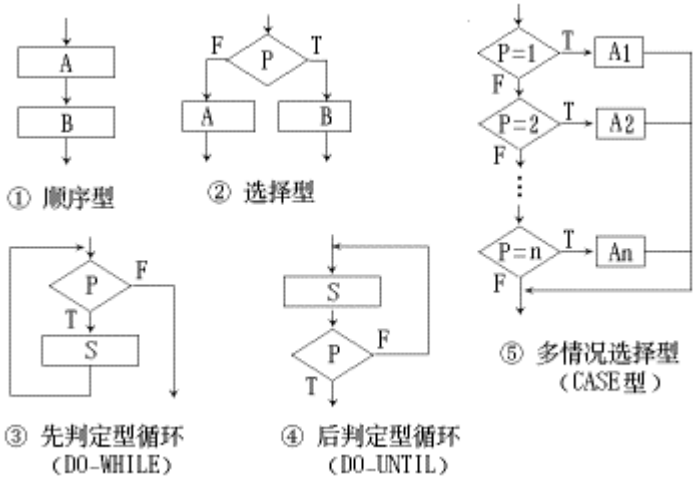


图 3.24 流程图的基本控制结构

任何复杂的程序流程图都应由这五种基本控制结构组匣蚯短锥　伞W魑　鲜鑫逯挚 刂平峁瓜嗷ㄗ楹虾颓短椎氖道　　?.25 给出一个程序的流程图。图中增加了一些虚线构成的框，目的是便于理解控制结构的嵌套关系。显然，这个流程图所描述的程序是结构化的。

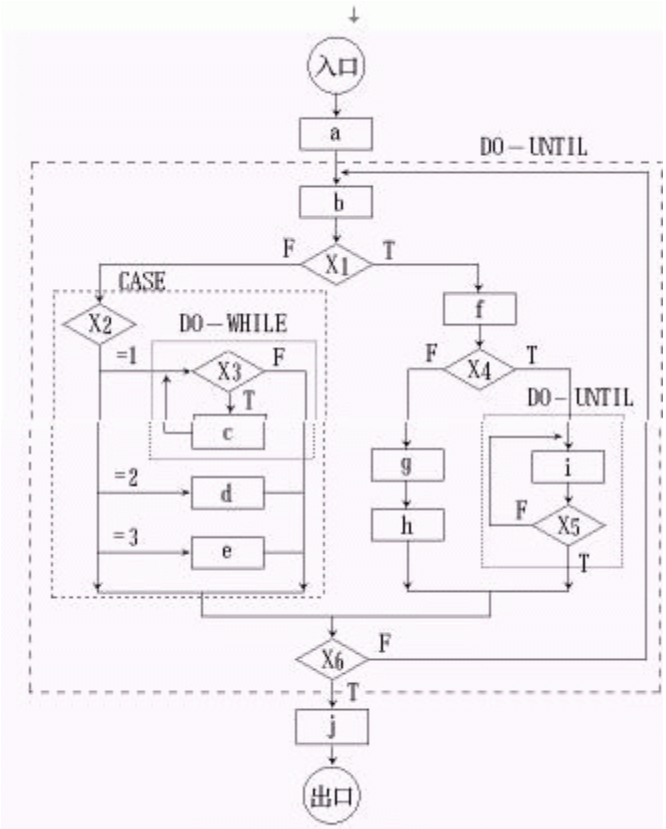


图 3.25 嵌套构成的流程图实例

(2) N-S 图

Nassi 和 Shneiderman 提出了一种符合结构化程序设计原则的图形描述工具，叫做盒图，也叫做 N-S 图。为表示五种基本控制结构，在 N-S 图中规定了五种图形构件。参看图 3.26。

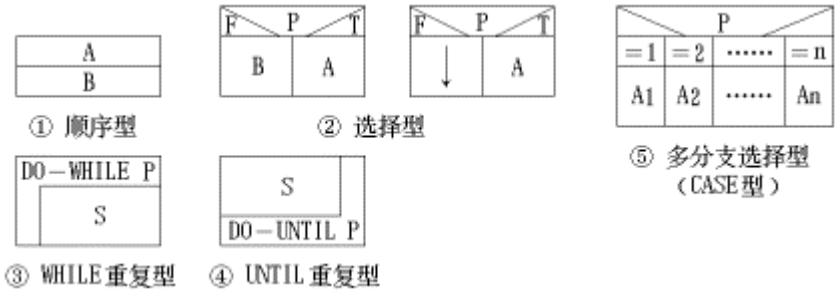


图 3.26 N-S 图的五种基本控制结构

为说明 N-S 图的使用，仍用图 3.25 给出的实例，将它用如图 3.27 所示的 N-S 图表示。

如前所述，任何一个 N-S 图，都是前面介绍的五种基本控制结构相互组合与嵌套的结果。当问题很复杂时，N-S 图可能很大。

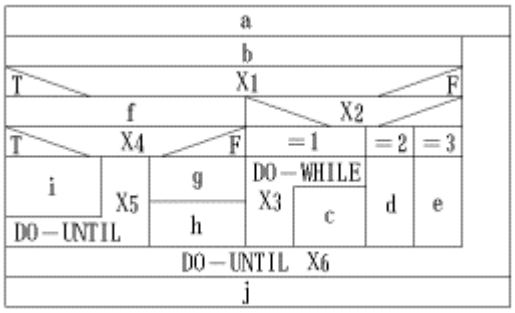


图 3.27 N-S 图的实例

(3) PAD

PAD 是 Problem Analysis Diagram 的缩写，它是日本日立公司提出，由程序流程图演化来的，用结构化程序设计思想表现程序逻辑结构的图形工具。现在已为 ISO 认可。

PAD 也设置了五种基本控制结构的图式，并允许递归使用。

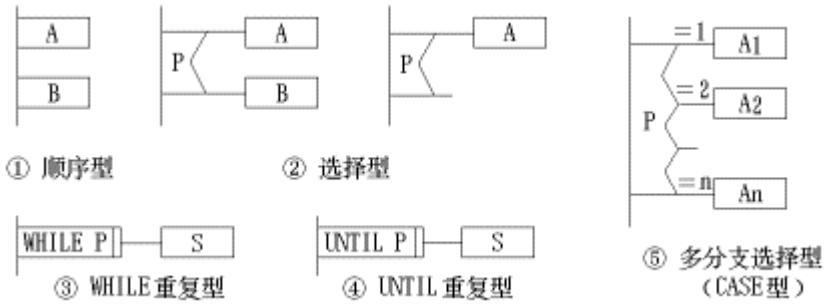


图 3.28 PAD 的基本控制结构

做为 PAD 应用的实例，图 3.29 给出了图 3.25 程序的 PAD 表示。PAD 所描述程序的层次关系表现在纵线上。每条纵线表示了一个层次。把 PAD 图从左到右展开。随着程序层次的增加，PAD 逐渐向右展开。

PAD 的执行顺序从最左主干线的上端的结点开始，自上而下依次执行。 每遇到判断或循环，就自左而右进入下一层，从表示下一层的纵线上端开始执行，直到该纵线下端，再返回上一层的纵线的转入处。如此继续，直到执行到主干线的下端为止。

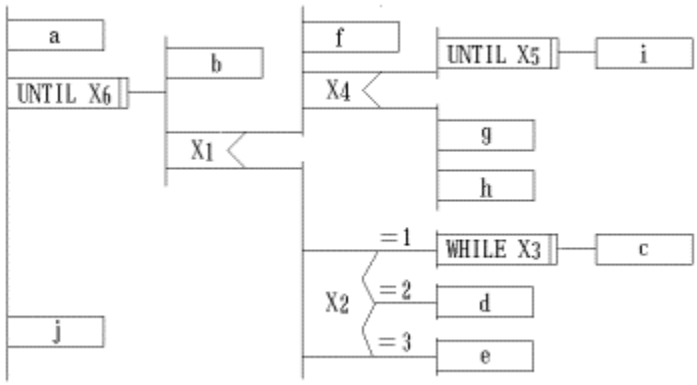


图 3.29 PAD 实例

(4) 判定表

当算法中包含多重嵌套的条件选择时，用程序流程图、N-S 图或 PAD 都不易清楚地描述。然而，判定表却能清晰地表达复杂的条件组合与应做动作之间的对应关系。仍然使用图 4.16 的例子。为了能适应判定表条件取值只能是"T"和"F"的情形，对原图稍微做了些改动，把多分支判断改为两分支判断，但整个图逻辑没有改变。见图 3.30。

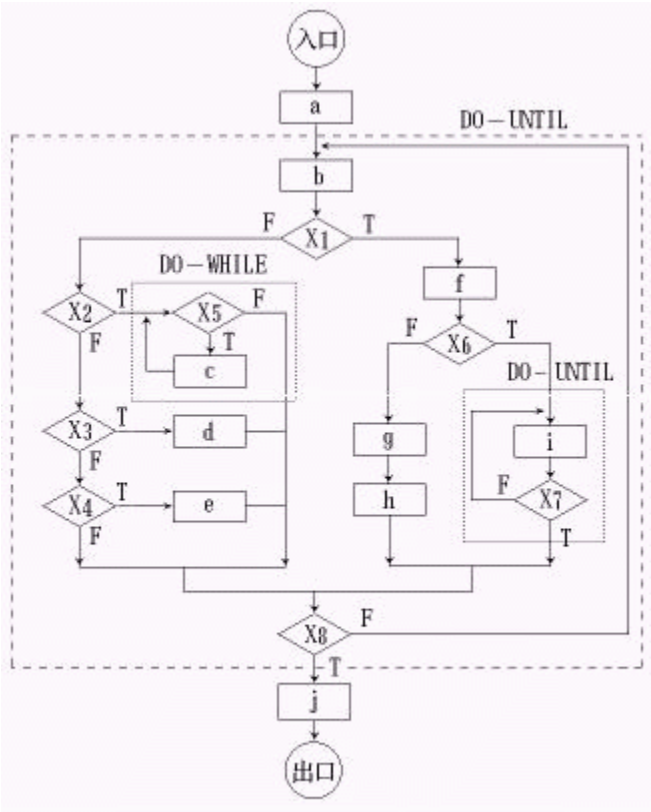


图 3.30 不包含多分支结构的流程图实例

与图 3.30 表示的流程图对应的判定表如图 3.31 所示。在表的右上半部分中列出所有条件，"T"表示该条件取值为真，"F"表示该条件取值为假，空白表示这个条件无论取何值对动作的选择不产生影响。在判定表右下半部分中列出所有的处

理，画"Y"表示要做这个动作，空白表示不做这个动作。判定表右半部的每一列实质上是一条规则，规定了与特定条件取值组合相对应的动作。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X1	T	T	T	T	T	F	F	F	F	F	F	F	F	F
X2	–	–	–	–	–	T	T	T	F	F	F	F	F	F
X3	–	–	–	–	–	–	–	–	F	F	T	T	F	F
X4	–	–	–	–	–	–	–	–	F	F	–	–	T	T
X5	–	–	–	–	–	T	F	F	–	–	–	–	–	–
X6	T	T	T	F	F	–	–	–	–	–	–	–	–	–
X7	T	T	F	–	–	–	–	–	–	–	–	–	–	–
X8	T	F	–	T	F	–	T	F	F	T	T	F	T	F
a	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
b	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
c	–	–	–	–	–	Y	–	–	–	–	–	–	–	–
d	–	–	–	–	–	–	–	–	–	–	Y	Y	–	–
e	–	–	–	–	–	–	–	–	–	–	–	–	Y	Y
f	Y	Y	Y	Y	Y	–	–	–	–	–	–	–	–	–
g	–	–	–	Y	Y	–	–	–	–	–	–	–	–	–
h	–	–	–	Y	Y	–	–	–	–	–	–	–	–	–
i	Y	Y	Y	–	–	–	–	–	–	–	–	–	–	–
j	Y	–	–	Y	–	–	Y	–	–	Y	Y	–	Y	–

图 3.31 反映程序逻辑的判定表

判定表的优点是能够简洁，无二义性地描述所有的处理规则。但判定表表示的是静态逻辑，是在某种条件取值组合情况下可能的结果，它不能表达加工的顺序，也不能表达循环结构，因此判定表不能成为一种通用的设计工具。

(5) PDL (Program Design Language)

PDL 是一种用于描述功能模块的算法设计和加工细节的语言。称为设计程序用语言。它是一种伪码。一般地，伪码的语法规则分为"外语法"和"内语法"。外语法应当符合一般程序设计语言常用语句的语法规则;而内语法可以用英语中一些简单的句子、短语和通用的数学符号，来描述程序应执行的功能。

PDL 就是这样一种伪码。它具有严格的关键字外语法，用于定义控制结构和数据结构，同时它的表示实际操作和条件的内语法又是灵活自由的，可使用自然语言的词汇。下面举一个例子，来看 PDL 的使用。

```
   PROCEDURE spellcheck IS 查找错拼的单词
BEGIN
split document into single words 把整个文档分离成单词
lood up words in dictionary 在字典中查这些单词
display words which are not in dictionary 显示字典中查不到的单词
create a new dictionary 造一新字典
END spellcheck
```

从上例可以看到，PDL 语言具有正文格式，很像一个高级语言。人们可以很方便地使用计算机完成 PDL 的书写和编辑工作。

PDL 作为一种用于描述程序逻辑设计的语言，具有以下特点：

§ 有固定的关键字外语法，提供全部结构化控制结构、数据说明和模块特征。

属于外语法的关键字是有限的词汇集，它们能对 PDL 正文进行结构分割，使 涞靡子诶斫狻N 饲 鹂、 郑 娑ü、 忠宦纱笮矗 渌 ゴ室宦尚へ础?br>

§ 内语法使用自然语言来描述处理特性。内语法比较灵活，只要写清楚就可以，不必考虑语法错，以利于人们可把主要精力放在描述算法的逻辑上。
§ 有数据说明机制，包括简单的（如标量和数组）与复杂的（如链表和层次结构）的数据结构。
§ 有子程序定义与调用机制，用以表达各种方式的接口说明。
使用 PDL 语言，可以做到逐步求精：从比较概括和抽象的 PDL 程序起，逐步写出更详细的更精确的描述。
(6) HIPO 图 （ Hierarchy plus Input Process Output ）
HIPO 最初只用做文档编写的格式要求，随后发展成比较有名的软件设计手段。HIPO 图采用功能框图和 PDL 来描述程序逻辑，它由两部分组成：可视目录表和 IPO 图。可视目录表给出程序的层次关系，IPO 图则为程序各部分提供具体的工作细节。
① 可视目录表 ：由体系框图、图例、描述说明三部分组成。
§ 体系框图。又称层次图（H 图），是可视目录表的主体，用它表明各个功能的隶属关系。它是自顶向下逐层分解得到的，是一个树形结构。它的顶层是整个系统的名称和系统的概括功能说明；第二层把系统的功能展开，分成了几个框；第二层功能进一步分解，就得到了第三层、第四层，…，直到最后一层。每个框内都应有一个名字，用以标识它的功能。还应有一个编号，以记录它所在的层次及在该层次的位置。
§ 图例。每一套 HIPO 图都应当有一个图例，即图形符号说明。附上图例，不管人们在什么时侯阅读它都能对其符号的意义一目了然。
§ 描述说明。它是对层次图中每一框的补充说明，在必须说明时才用，所以它是可选的。描述说明可以使用自然语言。
例如，应用 HIPO 法对盘存 / 销售系统进行分析。得到如图 3.32 所示的工作流程图。

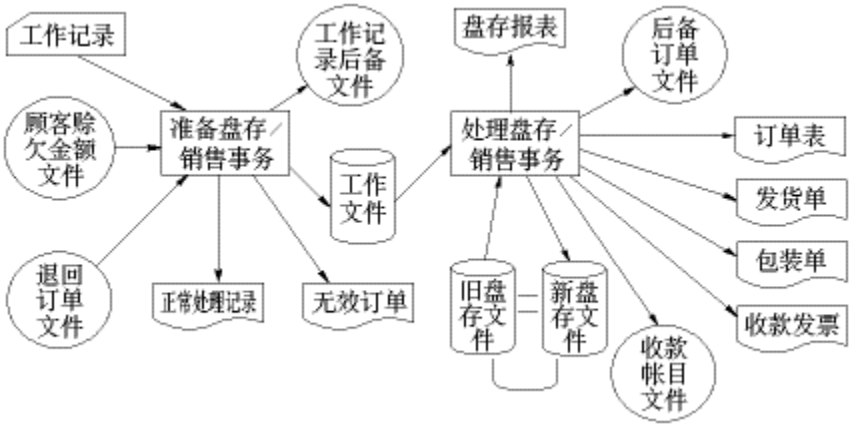
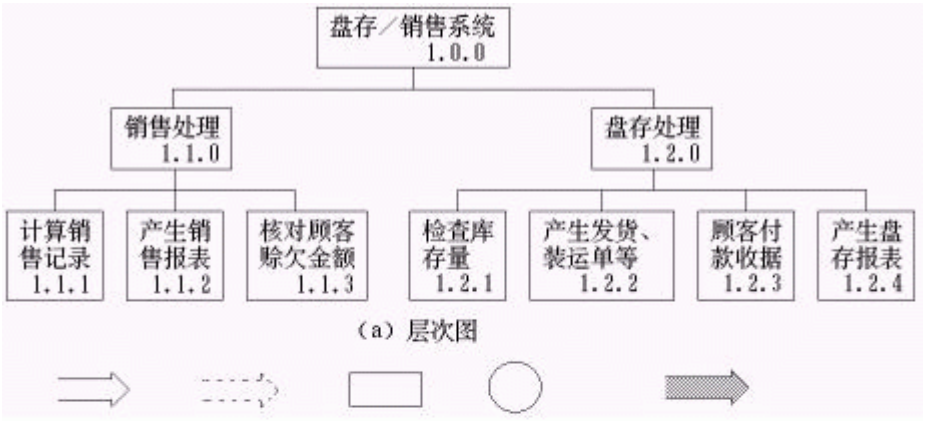


图 3.32 盘存 / 销售系统工作流程图

分析此工作流程图，可得如图 3.33 所示的可视目录表。图 3.33(a)是系统的层次图，图 3.33(b)是后面 IPO 图的图例，图 3.33(c)是描述说明。
② IPO 图 ：IPO 图为层次图中每一功能框详细地指明输入、处理及输出。通常，IPO 图有固定的格式，图中处理操作部分总是列在中间，输入和输出部分分别在其左边和右边。由于某些细节很难在一张 IPO 图中表达清楚，常常把 IPO 图又分

为两部分，简单概括的称为概要 IPO 图，细致具体一些的称为详细 IPO 图。



号 码	说 明
1.0.0	销售／盘存系统框图：处理订货、维护盘存文件、产生发货单、包装单、货运单、顾客付款收据、产生盘存与销售报表
1.1.0	顾客订单检查、核对顾客赊欠金额、产生销售报表
1.1.1	用工作文件的盘存项目号，对顾客订单进行核对和排序
1.1.2	以地区和人员为单位，编制销售报表，计算销售佣金
1.1.3	检验顾客赊欠金额，计算折扣，确定支付项目
1.2.0	处理盘存管理报表，顾客付款收帐，处理发货、包装、托运
⋮	

(c) 描述说明

图 4.33 盘存／销售系统的可视目录表

概要 IPO 图用于表达对一个系统，或对其中某一个子系统功能的概略表达，指明在完成某一功能框规定的功能时需要哪些输入，哪些操作和哪些输出。图 3.34 是表示销售／盘存系统第二层的对应于 H 图上的 1.1.0 框的概要 IPO 图。

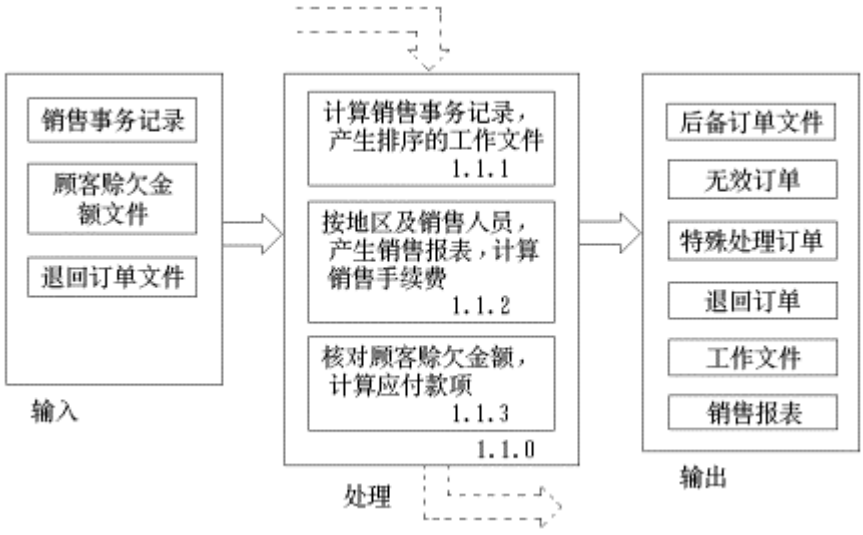


图 3.34 对应 H 图上 1.1.0 框的概要 IPO 图

在概要 IPO 图中，没有指明输入－处理－输出三者之间的关系， 用它来进行下一步的设计是不可能的。故需要使用详细 IPO 图以指明输入－处理－输出三者之间的关系，其图形与概要 IPO 图一样，但输入、输出最好用具体的介质和设备类型的图形表示。图 3.35 是销售 / 盘存系统中对应于 1.1.2 框的一张详细 IPO 图。

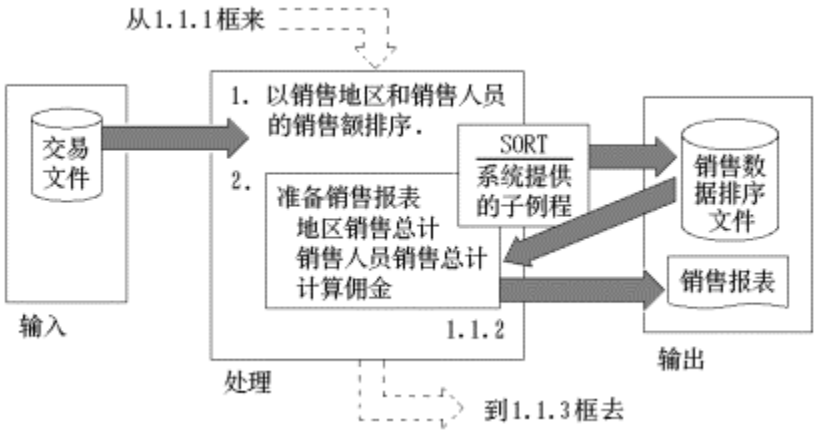


图 3.35 对应于 H 图 1.1.2 框的详细 IPO 图

③ 利用 HIPO 进行迭代式细化设计

在软件设计时，解决设计问题通常需要经历一个认识逐步发展的过程，并且对一些问题还要经过反复的考虑才可能达到比较满意的设计效果。我们称此为迭代式细化设计。HIPO 能很好地适应这一要求。图 3.36 是利用 HIPO 进行迭代式细化设计的示意图。从图中可看到，把可视目录表和 IPO 图结合起来，反复交替地使用它们，可使得设计工作逐步深化，最终取得完满的设计结果。其实这正是自顶向下，逐步求精的结构化程序设计思想。

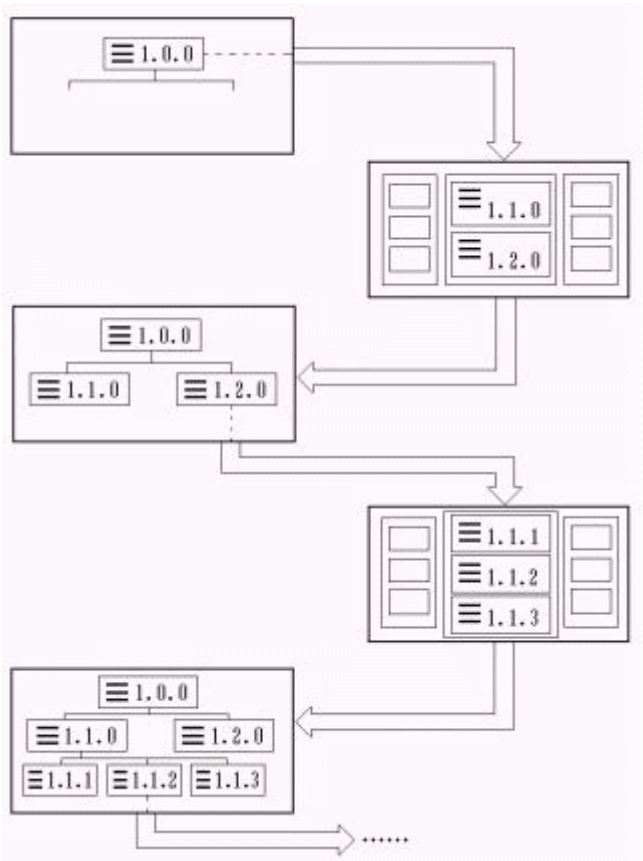


图 3.36 利用 HIPO 进行迭代式细化设计

HIPO 有自己的特点。首先，这一图形表达方法容易看懂。其次，HIPO 的适用范围很广，绝不限于详细设计。事实上，画可视目录表就是与概要设计密切相关的工作。如果利用它仅仅表达软件要达到的功能，则是需求分析中描述需求的很好的工具。因为 HIPO 是在开发过程中的表达工具，所以它又是开发文档的编制工具。开发完成后，HIPO 图就是很好的文档，而不必在设计完成以后，专门补写文档。

9. 设计规格说明与设计评审

软件设计规格说明的大纲如表 3.3 所示。每一个编号的段落描述了设计模型的不同侧面。在设计人员细化他们的软件设计时，就可以逐步完成各章节内容的嘈础?/p>

表 3.3 软件设计规格说明的大纲

I. 工作范围 A. 系统目标 B. 运行环境 C. 主要软件需求 D. 设计约束／限制
II. 体系结构设计 A. 数据流与控制流复审 B. 导出的程序结构 C. 功能与程序交叉索引
III. 数据设计 A. 数据对象与形成的数据结构 B. 文件和数据库结构 i 文件的逻辑结构 ii 文件逻辑记录描述 iii 访问方式 C. 全局数据 D. 文件／数据与程序交叉索引
IV. 接口设计 A. 人机界面规格说明 B. 人机界面设计规则 C. 外部接口设计 i 外部数据接口 ii 外部系统或设备接口 D. 内部接口设计规则
V.（每个模块的）过程设计 A. 处理与算法描述 B. 接口描述 C. 设计语言(或其它)描述 D. 使用的模块 E. 内部程序逻辑描述 F. 注释／约束／限制
VI. 运行设计 A. 运行模块组合 B. 运行控制规则 C. 运行时间安排
VII. 出错处理设计 A. 出错处理信息 B. 出错处理对策 i 设置后备 ii 性能降级 iii 恢复和再启动
VIII. 安全保密设计
IX. 需求／设计交叉索引
X. 测试部分 A. 测试方针 B. 集成策略 C. 特殊考虑
XI. 特殊注解
XII. 附录

软件设计的最终目标是要取得最佳方案。"最佳"是指在所有候选方案中，就节省开发费用，降低资源消耗，缩短开发时间的条件，选择能够赢得较高的生产率、较高的可靠性和可维护性的方案。在整个设计的过程中，各个时期的设计结果需要经过一系列的设计质量的评审，以便及时发现和及时解决在软件设计中出现的问题，防止把问题遗留到开发的后期阶段，造成后患。

设计评审的内容包括：

- 可追溯性：即分析该软件的系统结构、子系统结构，确认该软件设计是否复盖了所有已确定的软件需求，软件每一成分是否可追溯到某一项需求。
- 接口：即分析软件各部分之间的联系，确认该软件的内部接口与外部接口是否已经明确定义。模块是否满足高内聚和低耦合的要求。模块作用范围是否在其控制范围之内。
- 风险：即确认该软件设计在现有技术条件下和预算范围内是否能按时实现。
- 实用性：即确认该软件设计对于需求的解决方案是否实用。
- 技术清晰度：即确认该软件设计是否以一种易于翻译成代码的形式表达。
- 可维护性：从软件维护的角度出发，确认该软件设计是否考虑了方便未来的维护。
- 质量：即确认该软件设计是否表现出良好的质量特征。
- 各种选择方案：看是否考虑过其它方案，比较各种选择方案的标准是什么。
- 限制：评估对该软件的限制是否现实，是否与需求一致。
- 其它具体问题：对于文档、可测试性、设计过程，……，等等进行评估。

在这里需要特别注意：软件系统的一些外部特性的设计，例如软件的功能、一部分性能、以及用户的使用特性等，在软件需求分析阶段就已经开始。这些问题的解决，多少带有一些"怎么做"的性质，因此有人称之为软件的外部设计。

三、例题分析

【例 1】从下列有关系统结构图的叙述中选出正确的叙述。
(1) 系统结构图中反映的是程序中数据流的情况。
(2) 系统结构图是精确表达程序结构的图形表示法。因此，有时也可将系统结构当作程序流程图使用。
(3) 一个模块的多个下属模块在系统结构图中所处的左右位置是无关紧要的。
(4) 在系统结构图中，上级模块与其下属模块之间的调用关系用有向线段表示。这时，使用斜的线段和水平、垂直的线段具有相同的含义。
答案： (4)
分析：系统结构图反映的是系统中模块的调用关系和层次关系，谁调用谁，有一个先后次序（时序）关系。所以系统结构图既不同于数据流图，也不同与程序流程图。数据流图仅描述数据在系统中如何流动，如何处理和存储，它不考虑时序关系。图中的有向线段表示了数据流。程序流程图描述程序中控制流的情况，即程序中处理的执行顺序和执行序列所依赖的条件，图中的有向线段（流线）表示的是控制流，从一个处理走到下一个处理。但在系统结构图中的有向线段表示调用时程序的控制从调用模块移到被调用模块，并隐含了当调用结束时控制将交回给调用模块。
如果一个模块有多个下属模块，这些下属模块的左右位置可能与它们的调用次序有关。例如，在用结构化设计方法依据数据流图建立起来的变换型系统结构图中，主模块的所有下属模块按逻辑输入、中心变换、逻辑输出的次序自左向右一字排开，左右位置不是无关紧要的。所以只有最后的一个叙述是正确的。
【例 2】软件的开发工作经过需求分析阶段，进入(A)以后，就开始着手解决"怎么做"的问题。常用的软件设计方法有(B)、(C)、(D)和(E)等方法。
供选择的答案：
A ~ B. ① 程序设计 ② 设计阶段 ③ 总体设计 ④ 定义阶段
⑤ SD 方法 ⑥ SP 方法
C. ① Jackson 方法 ② 瀑布法 ③ 快速原型法 ④ 回溯法
D ~ E. ① LCP（Wanier）方法 ② 递归法 ③ Parnas 方法
④ 自下而上修正 ⑤ 逐步求精法 ⑥ 检测校正法
答案：A. ②, B. ⑤, C. ①, D. ①, E. ③。其中，D 与 E 的答案可互换。
分析：进入设计阶段之后，就开始着手解决"怎么做"的问题。一般把设计阶段的工作分成两步：即概要设计和详细设计。在概要设计阶段应着重解决实现需求的程序模块划分问题，在详细设计阶段则要决定每个模块的具体算法。
常见的软件概要设计方法有三大类：
§ 以数据流图为基础构造模块结构的结构化设计方法（SD）；
§ 以数据结构为基础构造模块结构的 Jackson 方法和 LCP（Wanier）逻辑构造方法；
§ 以对象、类、继承和通信为基础的面向对象设计方法（OOD）。
此外，以信息隐蔽为原则的 Parnas 方法虽然没有给出系统化的设计方法，但它提出了一组原则，要求预先估计未来生存周期中可能发生的种种情况，并采取相应措施以提高软件系统的可维护性和可靠性。
这里对面向数据结构的 Jackson 方法和 LCP 方法再多说几句。
Jackson 方法是一种典型的面向面向数据结构开发软件的方法。它的基本思想是首先根据实际问题，给出处理问题所需要和产生的数据结构，一旦搞清了问题的输入／输出数据结构，就可以以简单的方式直接导出程序的处理结构，然后应用

Jackson 的描述符号，将这个处理结构转换为程序的过程性描述。
LCP 方法是另一种面向数据结构的方法，它也要先给出用 Wanier 图表示的处理问题所需要和产生的数据结构，再在 Wanier 图上直接将数据结构转换为加工处理的形式化表示，最后生成描述加工过程的伪代码，进行验证和优化。
【例 3】请将下述有关模块独立性的各种模块之间的耦合，按其耦合度从低到高排列起来。
① 内容耦合 ② 控制耦合 ③ 非直接耦合 ④ 标记耦合
⑤ 数据耦合 ⑥ 外部耦合 ⑦ 公共耦合
答案：③、⑤、④、②、⑥、⑦、①
分析：参看"内容提要"中有关模块独立性的介绍。
【例 4】请将下述有关模块独立性的各种模块内聚，按其内聚度（强度）从高到低排列起来。
① 巧合内聚 ② 时间内聚 ③ 功能内聚 ④ 通信内聚
⑤ 逻辑内聚 ⑥ 信息内聚 ⑦ 过程内聚
答案： ③、⑥、④、⑦、②、⑤、①
分析：在状态－迁移图中，由一个状态和一个事件所确定的下一状态可能会有多个。实际会迁移到哪一个状态，是由更详细的内部状态和更详细的事件信息来决定的，此时在状态－迁移图中可能需要使用加进判断框和处理框的记法。状态－迁移图的优点：第一，状态之间的关系能够直观地捕捉到，这样用眼睛就能看到是否所有可能的状态迁移都已纳入图中，是否存在不必要的状态等。第二，由于状态－迁移图的单纯性，能够机械地分析许多情况，可很容易地建立分析工具。
【例 5】在结构化分析方法中，用实体－关系图表达系统中的对象及其关系。在实体－关系图中，表达对象的实例之间的关联有三种类型： 一对一联系、（ ）联系、多对多联系。
供选择的答案：
A. 多对一 B. 一对多
答案：B
分析：使用实体－关系图，可以建立系统中各个数据对象及对象之间的关系。对象的实例间的关联称为"基数"，共有 3 种类型的基数：一对一，一对多，多对多。它反映了现实世界中实体之间的联系，多对一的情况可以归入一对多的关联中去。
【例 6】 软件需求分析的任务不应包括（ A ）。进行需求分析可使用多种工具，但（ B ）是不适用的。在需求分析中，分析员要从用户那里解决的最重要的问题是（ C ）。需求规格说明书的内容不应当包括（ D ）。该文档在软件开发中具有重要的作用，但其作用不应当包括（ E ）。
供选择的答案：
A. ① 问题分析 ② 信息域分析 ③ 结构化程序设计 ④ 确定逻辑模型
B. ① 数据流图 ② 判定表 ③ PAD 图 ④ 数据词典
C. ① 要让软件做什么 ② 要给该软件提供哪些信息
③ 要求软件工作效率如何 ④ 要让软件具有什么样的结构
D. ① 对重要功能的描述 ② 对算法的详细过程性描述
③ 软件确认准则 ④ 软件的性能
E. ① 软件设计的依据 ② 用户和开发人员对软件要"做什么"的共同理解
③ 软件验收的依据 ④ 软件可行性分析的依据

答案：A. ③ B. ③ C. ① D. ② E. ④
分析：软件需求分析的任务是通过与用户的合作，了解用户对待开发系统的要求；根据对用户要求的系统所在的信息域的调查、分析，确定系统的逻辑模型；并对求解的问题做适当的分解，使之适合于计算机求解。需求分析的结果是软件需求规格说明书。
结构化程序设计是在详细设计和编码阶段所采用的技术，而不是需求分析阶段要采用的技术。在需求分析阶段，分析人员可以用数据流图描述系统的数据流的变换和流向，用数据词典定义在数据流图中出现的数据流、数据文件、加工或处理，用判定表表示复杂条件和动作组合的情况。但 PAD 图是在详细设计阶段使用的描述加工逻辑的工具，不适用于需求分析。此外，软件需求分析阶段只确定软件系统要"做什么"，完成对重要功能、性能、确认准则的描述，至于"怎么做"由后续的设计阶段完成，对算法的详细过程性描述也是在设计阶段给出。软件可行性分析应在需求分析之前，所以需求分析规格说明不能成为可行性分析的依据。
【例 7】原型化方法是用户和软件开发人员之间进行的一种交互过程，适用于（A）系统。它从用户界面的开发入手，首先形成（ B ），用户（ C ），并就（ D ）提出意见，它是一种（ E ）型的设计过程。
供选择的答案：
A. ① 需求不确定性高的 ② 需求确定的 ③ 管理信息 ④ 决策支持
B. ① 用户界面使用手册 ② 用户界面需求分析说明书
③ 系统界面原型 ④ 完善的用户界面
C. ① 改进用户界面的设计 ② 阅读文档资料
③ 模拟用户界面的运行 ④ 运行用户界面原型
D. ① 同意什么和不同意什么 ② 使用和不使用哪一种编程语言
③ 程序的结构 ④ 执行速度是否满足要求
E. ① 自外向内 ② 自顶向下 ③ 自内向外 ④ 自底向上
答案：A. ①, B. ③, C. ④, D. ①, E. ①
分析：通常，原型是指模拟某种产品的原始模型。在软件开发中，原型是软件的一个早期可运行的版本，它反映最终系统的部分重要特性。
使用原型的原型化方法特别适用于需求不确定性较高的软件系统的开发。它的基本思想是根据用户给出的基本需求，通过快速实现构造出一个小型的可执行的模型，满足用户的基本要求，这就是系统界面原型。让用户计算机上实际运行这个用户界面原型，在试用的过程中得到亲身感受和受到启发，做出反应和评价，提出同意什么和不同意什么。然后开发者根据用户的意见对原型加以改进。随着不断试验、纠错、使用、评价和修改，获得新的原型版本，如此周而复始，逐步减少分析和通信中的误解，弥补不足之处，进一步确定各种需求细节，适应需求的变更，从而提高了最终产品的质量。它是一种自外向内型的设计过程。

四、习题

【3-1】逐步求精、分层过程与抽象等概念之间的相互关系如何？
【3-2】完成良好的软件设计应遵循哪些原则？
【3-3】如何理解模块独立性？用什么指标来衡量模块独立性？
【3-4】模块独立性与信息隐蔽（反映模块化有效程度的属性）有何关系？
【3-5】模块的内聚性程度与该模块在分层结构中的位置有关系吗？说明你的论

据。
【3-6】耦合性的概念和软件的可移植性有什么关系？请举例说明你的论述。
【3-7】从供选择的答案中选出正确的答案填入下列叙述中的（ ） 内。
模块内聚性用于衡量模块内部各成份之间彼此结合的紧密程度。
(1) 一组语句在程序中多处出现，为了节省内存空间把这些语句放在一个模块中，该模块的内聚性是（ A ）的。
(2) 将几个逻辑上相似的成分放在同一个模块中，通过模块入口处的一个判断决定执行哪一个功能。该模块的内聚性是（ B ）的。
(3) 模块中所有成分引用共同的数据，该模块的内聚性是（ C ）的。
(4) 模块内的某成份的输出是另一些成份的输入，该模块的内聚性是（ D ）的。
(5) 模块中所有成份结合起来完全一项任务，该模块的内聚性是（ E ）的。它具有简明的外部界面，由它构成的软件易于理解、测试和维护。
供选择的答案：
A ~ E： ① 功能内聚 ② 信息内聚 ③ 通信内聚 ④ 过程内聚
⑤ 巧合内聚 ⑥ 时间内聚 ⑦ 逻辑内聚
【3-8】从供选择的答案中选出正确的答案填入下面的（ ）中。
块间联系和块内联系是评价程序模块结构质量的重要标准。联系的方式、共用信息的作用、共用信息的数量和接口的（ A ）等因素决定了块间联系的大小。在块内联系中，（ B ）的块内联系最强。
SD 方法的总的原则是使每个模块执行（ C ）功能，模块间传送（ D ）参数，模块通过（ E ）语句调用其它模块，而且模块间传送的参数应尽量（ F ）。此外，SD 方法还提出了判定的作用范围和模块的控制范围等概念。SD 方法认为，（ G ）应该是（ H ）的子集。
供选择的答案：
A： ① 友好性 ② 健壮性 ③ 简单性 ④ 安全性
B： ① 巧合内聚 ② 功能内聚 ③ 通信内聚 ④ 信息内聚
C： ① 一个 ② 多个
D： ① 数据型 ② 控制型 ③ 混合型
E： ① 直接引用 ② 标准调用 ③ 中断 ④ 宏调用
F： ① 少 ② 多
G ~ H：① 作用范围 ② 控制范围
【3-9】从供选择的答案中选出应该填入下列关于软件设计的叙述的（ ）内的正确答案。
在众多的设计方法中，SD 方法是最受人注意的，也是最广泛应用的一种，这种方法可以同分析阶段的（ A ）方法及编程阶段的（ B ）方法前后衔接，SD 方法是考虑如何建立一个结构良好的程序结构，它提出了评价模块结构质量的两个具体标准--块间联系和块内联系。SD 方法的最终目标是（ C ），用于表示模块间调用关系的图叫（ D ）。
另一种比较著名的设计方法是以信息隐蔽为原则划分模块，这种方法叫（ E ）方法。
供选择的答案：
A ~ B：① Jackson ② SA ③ SC ④ Parnas ⑤ SP
C： ① 块间联系大，块内联系大 ② 块间联系大，块内联系小
③ 块间联系小，块内联系大 ④ 块间联系小，块内联系小

D： ① PAD ② HCP ③ SC ④ SADT ⑤ HIPO ⑥ NS
E： ① Jackson ② Parnas ③ Turing ④ Wirth ⑤ Dijkstra
【3-10】递归模块（即自己调用自己的模块）的概念如何能够与本章所介绍的设计原理与方法相适应？
【3-11】举例说明你对概要设计与详细设计的理解。有不需要概要设计的情况吗?
【3-12】软件详细设计工具可分为三类，即：图示工具、设计语言和表格工具。图示工具中，（ A ）简单而应用广泛、（ B ）表示法中，每一个处理过程用一个盒子表示，盒子可以嵌套。（ C ）可以纵横延伸，图形的空间效果好。
（ D ） 是一种设计和描述程序的语言，它是一种面向（ E ）的语言。
供选择的答案：
A ~ C：① NS 图 ② 流程图 ③ HIPO 图 ④ PAD 图
D： ① C ② PDL ③ RPOLOG ④ PASCAL
E： ① 人 ② 机器 ③ 数据结构 ④ 对象
【3-13】如何用 PDL 语言来实施逐步求精的设计原理?
【3-14】从供选择的答案中选出应该填入下列关于软件设计的叙述的（ ）内的正确答案。
在完成软件概要设计，并编写出相关文档之后，应当组织对概要设计工作的评审。评审的内容包括：
分析该软件的系统结构、子系统结构，确认该软件设计是否覆盖了所有已确定的软件需求，软件每一成份是否可（ A ）到某一项需求。分析软件各部分之间的联系，确认该软件的内部接口与外部接口是否已经明确定义。模块是否满足（ B ）和（ C ）的要求。模块（ D ）是否在其（ E ）之内。
供选择的答案
A： ① 覆盖 ② 演化 ③ 追溯 ④ 等同 ⑤ 连接
B： ① 多功能 ② 高内聚 ③ 高耦合 ④ 高效率 ⑤ 可读性
C： ① 多入口 ② 低内聚 ③ 低耦合 ④ 低复杂度 ⑤ 低强度
D ~ E：① 作用范围 ② 高内聚 ③ 低内聚 ④ 取值范围 ⑤ 控制范围 五、习题解答

【3-1】"自顶向下，逐步求精"是 Niklaus Wirth 提出的设计策略：即将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。
这样的结构实际就是一个模块的分层结构，即分层的过程。在实施时，采用抽象化的方法，自顶向下，给出不同的抽象层次。在最高的抽象层次上，可以使用问题所处环境的语言概括地描述问题的解法。而在较低的抽象层次上，则采用过程化的方法。在描述问题的解法时，我们可以配合使用面向问题的术语和面向现实的术语。但最后在最低的抽象层次上，我们应使用能够直接实现的方式来描述这个解法。
【3-2】软件设计既是过程又是模型。设计过程是一系列迭代的步骤，使设计人员能够描述被开发软件的方方面面。设计模型体现了自顶向下、逐步细化的思想，首先构造事物的整体，再逐步细化，引导人们构造各种细节。为了给软件设计人员提供一些指导，1995 年 Davis 提出了一系列软件设计的原则如下，其中有些修改和补充：
• 设计过程不应受"隧道视野"的限制。一位好的设计者应当考虑一些替代的手

段。根据问题的要求，可以用基本的设计概念，如抽象、逐步求精、模块化、软件体系结构、控制层次、结构分解、数据结构、软件过程、信息隐蔽等，来决定完成工作的资源。

• 设计应能追溯到分析模型。由于设计模型中每一单个成份常常可追溯到多个需求上，因此有必要对设计模型如何满足需求进行追踪。

• 设计不应当从头做起。系统是使用一系列设计模式构造起来的，很多模式很可能以前就遇到过。这些模式通常被称为可复用的设计构件。可以使用它们代替那些从头开始做的方法。时间短暂而资源有限！设计时间应当投入到表示真正的新思想和集成那些已有的设计模式上去。

• 设计应当缩短软件和现实世界中问题的"智力差距"，就是说，软件设计的结果应尽可能模拟问题领域的结构。

• 设计应具有一致性和集成性。如果一个设计从整体上看像是一个人完成的，那它就是一致的。在设计工作开始之前，设计小组应当定义风格和格式的规则，如果仔细定义了设计构件之间的接口，则该设计就是集成的。

• 使用上述的基本的设计概念，将设计构造得便于将来的修改。

• 应将设计构造得即使遇到异常的数据、事件或操作条件，也能平滑地、轻松地降级。设计良好的计算机程序应当永不"彻底停工"，它应能适应异常的条件，并且当它必须中止处理时也能以从容的方式结束。

• 设计不是编码，编码也不是设计。即使在建立程序构件的详细的过程设计时，设计模型的抽象级别也比源代码要高。在编码级别上作出的唯一设计决策是描述如何将过程性设计转换为程序代码的小的实现细节。

• 在开始着手设计时就应当能够评估质量，而不是在事情完成之后。利用上述的基本的设计概念和已有的设计方法，可以帮助设计者评估质量。

• 应当坚持设计评审以减少概念上（语义上）的错误。有时人们在设计评审时倾向于注重细节，只见树木不见森林。在关注设计模型的语法之前，设计者应能确保设计的主要概念上的成份（的遗漏、含糊、不一致）都已检查过。

【3-3】如果两个模块互相独立，那么对其中一个模块进行编码、测试或修改时可以完全不考虑另一个模块对它的影响。因此，用模块独立性作为衡量模块结构是否容易编码、容易测试、容易修改的标准是合适的。但是，在一个系统的模块结构中没有哪两个模块可以完全独立，所以，要力争模块之间尽量独立，以得到一个质量良好的模块结构。

一般采用两个准则度量模块独立性。即模块间的耦合和模块的内聚。模块间的耦合是模块之间的相对独立性（互相连接的紧密程度）的度量。模块之间的连接越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。内聚是模块功能强度（一个模块内部各个成份彼此结合的紧密程度）的度量。一个模块内部各个成份之间的联系越紧密，则它的内聚性就越高，相对地，它与其它模块之间的耦合性就会减低，而模块独立性就越强。因此，模块独立性比较强的模块应是高内聚低耦合的模块。

一般模块之间可能的连接方式有七种，构成耦合性的七种类型。它们之间的关系为

低耦合的情形有非直接耦合、数据耦合和标记耦合，它们都是比较好的模块间的连接。特点是模块间的接口简单、规范。中度耦合的情形有控制耦合，它通过参数表传递控制参数。相对高的耦合情形有外部耦合和公共耦合，它们都是通过全

局数据传递模块间的信息，不是说它们一定"坏"，但一定要注意使用这类耦合可能产生的后果，特别要防范这种后果。
一般模块的内聚性分为七种类型，它们的关系如下图所示。

在上面的关系中可以看到，位于高端的几种内聚类型最好，位于中段的几种内聚类型是可以接受的，但位于低端的内聚类型很不好，一般不能使用。因此，人们总是希望一个模块的内聚类型向高的方向靠。模块的内聚在系统的模块化设计中是一个关键的因素。
内聚和耦合是相互关联的。在程序结构中各模块的内聚程度越高，模块间的耦合程度就越低。但这也不是绝对的。我们的目标是力求增加模块的内聚，尽量减少模块间的耦合，但增加内聚比减少耦合更重要，应当把更多的注意力集中到提高模块的内聚程度上来。
【3-4】所谓"模块独立性"是指软件系统中每个模块只涉及软件要求的具体的子功能，而和软件系统中其它的模块的接口是简单的。所谓的"信息隐蔽"是指每个模块的实现细节对于其它模块来说是隐蔽的。也就是说，模块中所包含的信息(包括数据和过程）不允许其它不需要这些信息的模块使用。
如果软件系统做到了信息隐蔽，即定义和实施了对模块的过程细节和局部数据结构的存取限制，那么这些模块相互间的接口就是简单的。这组模块的独立性就比较强。事实上，衡量模块独立性的一个准则就是模块内聚，达到信息隐蔽的模块是信息内聚模块，它是高内聚情形，模块独立性当然很强了。
一个对象的抽象数据类型，就是信息隐蔽的示例。例如，对于栈 stack， 可以定义它的操作 makenull（置空栈）、push（进栈）、pop（退栈）、gettop（取栈顶）和 empty（判栈空）。这些操作所依赖的数据结构是什么样的? 它们是如何实现的? 都被封装在其实现模块中。软件的其它部分可以直接使用这些操作，不必关心它的实现细节。一旦实现栈 stack 的模块里内部过程或局部数据结构发生改变，只要它相关操作的调用形式不变， 则软件中其它所有使用这个栈 stack 的部分都可以不修改。 这样的模块结构具有很强的模块独立性。
【3-5】模块的内聚性与该模块在分层模块结构中的位置无关。事实上，一个好的模块化的程序系统，它所有的模块可以都是功能内聚的，即每一个模块就只干了一件事。用结构化设计方法建立起来的模块结构中的每一个模块都符合这个要求。把讨论范围再拓宽点，在纯面向对象范型的软件系统中，整个系统看作是一个类，它的子类可以看作是系统的子系统或高层模块，它们还可以有子类，……，这就形成一个类的层次结构。类的构造可以看成是一个抽象数据类型，实际上是信息内聚的。所以整个系统中从上到下，所有模块（对象类）都是信息内聚的模块。
【3-6】所谓"耦合性"是指模块之间联系的紧密程度的一种度量，而软件的"可移植性"是指将一个软件系统从一个计算机系统或环境移植到另一个计算机系统或环境中运行时所需工作量的大小。可移植性是用一组子特性，包括简明性、模块独立性、通用性、可扩充性、硬件独立性和软件系统独立性等，来衡量的。如果一个软件具有可移植性，它必然耦合性低，这样模块独立性要强。例如，有一个图形处理软件，它应具有二维几何图形处理、三维几何图形处理、图形显示、外设控制、数据库管理、用户界面控制、设计分析等模块。如果这些模块之间都是通过参数表来传递信息，那么它们之间的的耦合就是数据耦合或标记耦合等，都是低耦合。将来如果想要把它们移植到另一个外部环境中，这些模块容易修改(功

能内聚），且接口清晰，修改可局部化。反言之，如果这些模块都是功能内聚或信息内聚的模块，模块之间的耦合都是低耦合，也对可移植性有促进。但不能讲具有低耦合性模块结构的软件一定具有可移植性，因为是否具有可移植性还有其它因素的影响。

【3-7】 A. ⑤, B. ⑦, C. ③, D. ④, E. ①

【3-8】 A. ③, B. ②, C. ①, D. ①, E. ②, F. ①, G. ①, H. ②

模块之间的耦合（块间联系）和模块的内聚（块内联系）是评价程序模块结构质量的重要标准。联系的方式、共用信息的作用、共用信息的数量和接口的简单性等因素决定了块间联系的大小。在块内联系中，以功能内聚模块的块内联系最强。SD 方法的总的原则是使每个模块只做一件事，就是说，只执行一个功能。模块之间尽可能传送简单的数据型参数。模块要通过标准调用语句调用其它模块，不要直接引用另一个模块内部的数据。同时模块之间传送的参数应尽量少。此外，SD 方法还提出了判定的作用范围和模块的控制范围等概念。SD 方法认为，模块的作用范围应该是其控制范围的子集。

【3-9】 A. ②, B. ⑤, C. ③, D. ③, E. ②

结构化设计方法（SD）是一种应用非常广泛的软件设计方法，它以结构化分析方法（SA）得到的数据流图和数据词典为依据，建立软件的模块结构，然后对每一个模块用结构化程序设计（SP）方法设计它的内部逻辑。这几种方法是前后衔接的。用 SD 方法建立的模块结构用模块间的耦合（块间联系）和模块的内聚（块内联系）来度量，要求一个好的模块结构应满足高内聚，低耦合。在 SD 方法中表示模块间调用关系的图叫做系统结构图（SC）。

另一种著名的设计方法是以信息隐蔽为原则划分模块，这种方法是 Parnas 提出来的，叫做 Parnas 方法。

【3-10】递归过程在求解复杂的大型问题时非常有效。常用的求解问题的方法，一种叫做"分而治之"的策略和"回溯"的策略，都可以用递归方法来解决。所谓"分而治之"的方法即是把大而复杂的问题化为规模稍小的子问题，用同样方法求解。如果分解后的子问题能够直接解决，就直接解出，然后再回推得到原来问题的解。所谓"回溯"方法就是如果一个大的问题在求解过程中从某一步出发有可选的多种解决方案，先选择一种解决方案，试探求解。如果求解失败，撤消原来的选择，再选一种解决方案，试探求解，……。如果用某一方案求解成功，则退回上一步并报告这一步求解成功；如果所有可选方案都试过，都求解失败，则退回上一步并报告求解失败。

软件设计过程中的"自顶向下，逐层分解"的做法与上述求解问题的策略是一致的。如果分解出来的子问题（子功能、子模块）相互独立性比较强，这种分解可以降低模块的复杂性，做到模块化。所以，只要分解出来的子问题与原来问题满足递归的情况，用递归方法建立模块结构也是可行的。

【3-11】软件设计是一个把软件需求变换成软件表示的过程。最初这种表示只是描绘出软件的总的框架，然后进一步细化，在此框架中填入细节，把它加工成在程序细节上非常接近于源程序的软件表示。正因为如此，所以从工程管理的角度来看，软件设计分两步完成。首先做概要设计，将软件需求转化为数据结构和软件的系统结构。然后是详细设计，即过程设计。通过对结构表示进行细化，得到软件的详细的数据结构和算法。

由于概要设计建立起整个系统的体系结构框架，并给出了系统中的全局数据结构和数据库接口，人机接口，与其它硬、软件的接口。此外还从系统全局的角度，

考虑处理方式、运行方式、容错方式、以及系统维护等方面的问题，并给出了度量和评价软件质量的方法，所以它奠定了整个系统实现的基础。没有概要设计，直接考虑程序设计，就不能从全局把握软件系统的结构和质量，实现活动处于一种无序状态，程序结构划分不合理，导致系统处于一种不稳定的状态，稍一做改动就会失败。所以，不能没有概要设计。

【3-12】A. ②, B. ①, C. ④, D. ②, E. ①

【3-13】使用 PDL 语言，可以做到逐步求精：从比较概括和抽象的 PDL 程序起，逐步写出更详细的更精确的描述。下面举一个例子。

```
   PROCEDURE spellcheck IS 查找错拼的单词
BEGIN
split document into single words 把整个文档分离成单词
lood up words in dictionary 在字典中查这些单词
display words which are not in dictionary 显示字典中查不到的单词
create a new dictionary 造一新字典
END spellcheck
```

这个例子只是搭起一个处理问题的框架。为进一步表明查找拼错的单词的 4 个步骤如何实现，可以对它每一步进行细化:

```
PROCEDURE spellcheck
BEGIN
--* split document into single words
LOOP get next word
add word to word list in sortorder
EXIT WHEN all words processed
END LOOP
--* look up words in dictionary
LOOP get word from word list
IF word not in dictionary THEN
--* display words not in dictionary
display word, prompt on user terminal
IF user response says word OK THEN
add word to good word list
ELSE
add word to bad word list
ENDIF
ENDIF
EXIT WHEN all words processed
END LOOP
--* create a new words dictionary
dictionary:＝merge dictionary and good word list
END spellcheck
```

【3-14】A. ③, B. ②, C. ③, D. ①, E. ⑤

第四章 程序编码

一、复习要求

1. 了解什么是结构化程序设计，以及结构化程序设计的原则。

2. 了解程序设计风格 4 个方面的要求。

3. 了解提高程序效率的方法。

4. 了解程序设计语言的分类和特点。

5. 掌握度量程序复杂性的 McCabe 方法和 Halstead 方法。

二、内容提要

1. 结构化程序设计

结构化程序设计技术是 60 年代中期提出来的，它主要包括两个方面：

§ 在编写程序时，强调使用几种基本控制结构，通过组合嵌套，形成程序的控制结构。尽可能避免使用会使程序质量受到影响的 GOTO 语句。

§ 在程序设计过程中，尽量采用自顶向下和逐步细化的原则，由粗到细，一步步展开。

(1) 结构化程序设计的原则

§ 使用语言中的顺序、选择、重复等有限的基本控制结构表示程序逻辑。

§ 选用的控制结构只准许有一个入口和一个出口。

§ 程序语句组成容易识别的块，每块只有一个入口和一个出口。

§ 复杂结构应该用基本控制结构进行组合嵌套来实现。

§ 语言中没有的控制结构，可用一段等价的程序段模拟，但要求该程序段在整个系统中应前后一致。

§ 严格控制 GOTO 语句,仅在用一个非结构化的程序设计语言去实现一个结构化的构造，或者在某种可以改善而不是损害程序可读性的情况下才可以使用 GOTO 语句。

大量采用 GOTO 语句实现控制路径，会使程序路径变得复杂而且混乱，因此要控制 GOTO 语句的使用。但有时完全不用 GOTO 语句进行程序编码，比用 GOTO 语句编出的程序可读性差。例如，在查找结束时，文件访问结束时，出现错误情况要从循环中转出时，使用布尔变量和条件结构来实现就不如用 GOTO 语句来得简洁易懂。

对于常用的高级程序设计语言，一般都具备前述的几种基本控制结构。即使不具备等同的结构，也可以采用仿真来实现。下面以 FORTRAN77 为例进行说明，参看图 4.1。

基本控制结构	用 FORTRAN77 模拟基本控制结构
判断语句 **if** (p) S1; **else** S2;	**IF (p) THEN** S1 **ELSE** S2 **ENDIF**
先判断型循环语句 **while** (p) S;	100 **CONTINUE** **IF (p) THEN** S **GOTO** 100 **ENDIF**
后判断型循环语句 **do** S; **while** (p);	100 **CONTINUE** S **IF (p) GOTO** 100

图 4.1 用 FORTRAN77 语句实现基本控制结构

(2) 程序设计自顶向下，逐步求精 在详细设计和编码阶段，应当采取自顶向下，逐步求精的方法，把一个模块的功能逐步分解，细化为一系列具体的步骤，进而翻译成一系列用某种程序设计语言写成的程序。

例如，要求用筛选法求 100 以内的素数。所谓筛选法，就是从 2 到 100 中去掉 2,3,…,9,10 的倍数，剩下的就是 100 以内的素数。为了解决这个问题，可先按程序功能写出一个框架。

```
main ( ) {
建立 2 到 100 的数组 A[ ]，其中 A[i]＝i； －－－－－－－－－－－－－－－－－－－－－－－－－－－ 1
建立 2 到 10 的素数表 B[ ]，其中存放 2 到 10 以内的素数；－－－－－－－－－－－－－ 2
若 A[i]＝i 是 B[ ]中任一数的倍数，则剔除 A[i]；－－－－－－－－－－－－－－－－－－－－－－ 3
输出 A[ ]中所有没有被剔除的数；－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－ 4
}
```

上述框架中每一个加工语句都可进一步细化成一个循环语句。

```
main ( ) {

/*建立 2 到 100 的数组 A[ ]，其中 A[i]＝i*/ －－－－－－－－－－－－－－－－－－－－－－－－－－－－1

for （i = 2；i <= 100；i++）A[i] = i；

/* 建立 2 到 10 的素数表 B[ ]，其中存放 2 到 10 以内的素数*/ －－－－－－－－－－－－2

B[1] =2； B[2] = 3； B[3] = 5； B[4] = 7；

/*若 A[i]＝i 是 B[ ]中任一数的倍数，则剔除 A[i]*/ －－－－－－－－－－－－－－－－－－－－－－3

for （j = 1； j <= 4； j++）

检查 A[ ]所有的数能否被 B[j]整除并将能被整除的数从 A[]中剔除；－－－－－－－－－3.1

/*输出 A[ ]中所有没有被剔除的数*/ －－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－4
for （i = 2； i <= 100； i++）
若 A[i]没有被剔除，则输出之；－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－4.1

}
```

继续对 3.1 和 4.1 细化下去，直到最后每一个语句都能直接用程序设计语言来表示为止。

```
main ( ) {

for （i = 2；i <= 100；i++）A[i] = i；

B[1] =2； B[2] = 3； B[3] = 5； B[4] = 7；

/*若 A[i]＝i 是 B[ ]中任一数的倍数，则剔除 A[i]*/

for （j = 1； j <= 4； j++）

/*检查 A[ ]所有的数能否被 B[j]整除并将能被整除的数从 A[ ]中剔除*/

for （i = 2； i <= 100； i++）

if （A[i] / B[j]*B[j] == A[i]）A[i] = 0；
```

```
/*输出 A[]中所有没有被剔除的数*/

for （i = 2； i <= 100； i++）

/*若 A[i]没有被剔除，则输出之*/

if （A[i] != 0）

printf（“A[%d]＝%d\n”，i，A[i]）；

}
```

自顶向下，逐步求精方法的优点：

① 自顶向下，逐步求精方法符合人们解决复杂问题的普遍规律。可提高软件开发的成功率和生产率；

② 用先全局后局部，先整体后细节，先抽象后具体的逐步求精的过程开发出来的程序具有清晰的层次结构，因此程序容易阅读和理

③ 程序自顶向下，逐步细化，分解成一个树形结构（如图 4.2 所示）。在同一层的结点上做的细化工作相互独立。在任何一步发生错误，一般只影响它下层的结点，同一层其它结点不受影响。在以后的测试中，也可以先独立地一个结点一个结点地做，最后再集成。

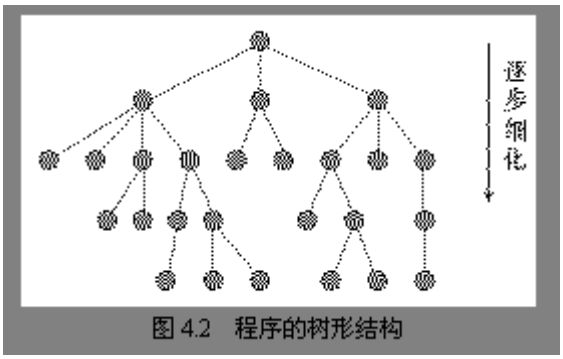


图 4.2 程序的树形结构

④ 程序清晰和模块化，使得在修改和重新设计一个软件时，可复用的代码量最大；

⑤ 每一步工作仅在上层结点的基础上做不多的设计扩展，便于检查；

⑥ 有利于设计的分工和组织工作。

(3) 数据结构的合理化

H.Mills 指出，结构化程序设计主要是想从程序的控制结构入手，消除不适应的、容易引起混乱的 GOTO 语句。这只是问题的一个方面，而问题的另一方面，过去没有注意到的是数据结构的合理化问题，即数据结构访问的规范化，标准化问题。

假如数据结构中常使用数组、指针等数据类型，则对它们必须采取随机访问，这样势必产生访问上的混乱。例如，要访问数组元素 A[i][j]，必须先对下标 i，j 访问，造成访问忽前忽后，这与 GOTO 语句造成的混乱类似，同样是有害的。
H.Mills 指出，解决这一问题的办法是用栈和队列去代替数组和指针。栈与队列分别是按后进先出（LIFO）和先进先出（FIFO）的原则进行存取的。在程序中用栈和队列代替数组和指针，用合理的规范的顺序存取代替随机存取，将克服随机存取带来的麻烦。而且有人做了证明，所有使用数组和指针的程序，都可以使用栈和队列的程序等价替换。

2．程序设计风格

在软件生存期中，人们经常要阅读程序。特别是在软件测试阶段和维护阶段，编写程序的人与参与测试、维护的人都要阅读程序。因此，阅读程序是软件开发和维护过程中的一个重要组成部分，而且读程序的时间比写程序的时间还要多。70 年代初，有人提出在编写程序时，应使程序具有良好的风格。

程序设计风格包括 4 个方面：源程序文档化，数据说明，语句结构和输入 / 输出方法，力图从编码原则的角度提高程序的可读性，改善程序质量。

(1) 源程序文档化

① 符号名的命名

符号名即标识符，包括模块名、变量名、常量名、子程序名、数据区名、缓冲区名等。 § 这些名字应能反映它所代表的实际东西，应有一定实际意义。

§ 名字不是越长越好，过长的名字会使程序的逻辑流程变得模糊，给修改带来困难。所以应当选择精炼的意义明确的名字，改善对程序功能的理解。

§ 必要时可使用缩写名字，但缩写规则要一致，并且要给每一个名字加注释。

§ 在一个程序中，一个变量只应用于一种用途。就是说，在同一个程序中一个变量不能身兼几种工作。

② 程序的注释

夹在程序中的注释是程序员与日后的程序读者之间通信的重要手段。正确的注释能够帮助读者理解程序，可为后续阶段进行测试和维护，提供明确的指导。因此，注释决不是可有可无的，大多数程序设计语言允许使用自然语言来写注释，这就给阅读程序带来很大的方便。一些正规的程序文本中，注释行的数量占到整个源程序的 1 / 3 到 1 / 2，甚至更多。

§ 序言性注释 ：通常置于每个程序模块的开头部分，它应当给出程序的整体说明，对于理解程序本身具有引导作用。有些软件开发部门对序言性注释做了明确而严格的规定，要求程序编制者逐项列出的有关项目包括：程序标题、有关本模块功能和目的的说明、主要算法、接口说明、有关数据描述、模块位置、开发简历等。

§ 功能性注释 ：嵌在源程序体中，用以描述其后的语句或程序段是在做什么工作，不要解释下面怎么做，因为解释怎么做常常是与程序本身重复的，并且对于阅读者理解程序没有什么帮助。

书写功能性注释，要注意：•用于描述一段程序，而不是每一个语句；•用缩进和空行，使程序与注释容易区别；•注释要正确。

③ 视觉组织

利用空格、空行和移行，提高程序的可视化程度。

§ 恰当地利用空格，可以突出运算的优先性，避免发生运算的错误。

§ 自然的程序段之间可用空行隔开；

§ 对于选择语句和循环语句，把其中的程序段语句向右做阶梯式移行。这样可使程序的逻辑结构更加清晰，层次更加分明。

(2) 数据说明

在编写程序时，需注意数据说明的风格。为了使程序中数据说明更易于理解和维护，必须注意以下几点。

§ 数据说明的次序应当规范化，使数据属性容易查找。

§ 当多个变量名用一个语句说明时，应当对这些变量按字母的顺序排列。

§ 如果设计了一个复杂的数据结构，应当使用注释来说明在程序实现时这个数据结构的固有特点。

(3) 语句结构

在设计阶段确定了软件的逻辑流结构，但构造单个语句则是编码阶段的任务。语句构造力求简单，直接，不能为了片面追求效率而使语句复杂化。 § 在一行内只写一条语句，并且采取适当的移行格式，使程序的逻辑和功能变得更加明确。

§ 程序编写首先应当考虑清晰性，不要刻意追求技巧性，使程序编写得过于紧凑。

§ 程序编写得要简单，写清楚，直截了当地说明程序员的用意。

§ 除非对效率有特殊的要求，程序编写要做到清晰第一，效率第二。不要为了追求效率而丧失了清晰性。事实上，程序效率的提高主要应通过选择高效的算法来实现。

§ 首先要保证程序正确，然后才要求提高速度。反过来说，在使程序高速运行时，首先要保证它是正确的。

§ 让编译程序做简单的优化。

§ 尽可能使用库函数。

§ 避免使用临时变量而使可读性下降。

§ 尽量用公共过程或子程序去代替重复的功能代码段。

§ 用调用公共函数去代替重复使用的表达式。

§ 使用括号来清晰地表达算术表达式和逻辑表达式的运算顺序。

§ 避免不必要的转移。同时如果能保持程序的可读性，则不必用 GOTO 语句。

§ 尽量只采用三种基本的控制结构来编写程序。

§ 用逻辑表达式代替分支嵌套。

§ 避免使用空的 ELSE 语句和 IF…THEN IF…的语句。

§ 避免使用 ELSE GOTO 和 ELSE RETURN 结构。

§ 使与判定相联系的动作尽可能地紧跟着判定。

§ 避免采用过于复杂的条件测试。

§ 尽量减少使用“否定”条件的条件语句。不要让读者绕弯子想。

§ 避免过多的循环嵌套和条件嵌套；

§ 不要使 GOTO 语句相互交叉。

§ 避免循环的多个出口。

§ 使用数组，以避免重复的控制序列。

§ 尽可能用通俗易懂的伪码来描述程序的流程，然后再翻译成必须使用的语言。

§ 数据结构要有利于程序的简化。

§ 要模块化，使模块功能尽可能单一化，模块间的耦合能够清晰可见。

§ 利用信息隐蔽，确保每一个模块的独立性。

§ 从数据出发去构造程序。

§ 不要修补不好的程序，要重新编写。也不要一味地追求代码的复用，要重新组织。

§ 对太大的程序，要分块编写、测试，然后再集成。

§ 对递归定义的数据结构尽量使用递归过程。

§ 注意计算机浮点数运算的特点，例如，浮点数运算 10.0*0.1 通常不等于 1.0 。

§ 不要单独进行浮点数的比较。用它们做比较，其结果常常发生异常情况。

§ 避免不恰当地追求程序效率，在改进效率前，要做出有关效率的定量估计。

§ 在程序中应有出错处理功能，一旦出现故障时不要让机器进行干预，导致停工。

此外，对于程序中的变量，标号，注释等，还需要给予一些注意。例如， § 变量名中尽量不用数字。

§ 显式说明所有的变量。

§ 确保所有变量在使用前都被初始化。

§ 确保注释与代码完全一致。

§ 不仅对代码做注释，而且对每条注释都加以编号。

§ 不注释不好的代码，要重新编写。

§ 程序格式的安排应有助于读者理解程序。

§ 注释不要过于繁琐。

§ 遵循国家标准。

§ 经常反躬自省：“如果我不是编码的人，我能看懂它吗？”考虑它的可理解性达到什么程度。

(4) 输入和输出 (I / 0)

输入和输出信息是与用户的使用直接相关的。输入和输出的方式和格式应当尽可能方便用户的使用。因此，在软件需求分析阶段和设计阶段，就应基本确定输入和输出的风格。系统能否被用户接受，有时就取决于输入和输出的风格。 不论是批处理的输入／输出方式，还是交互式的输入／输出方式，在设计和程序编码时都应考虑下列原则：

§ 对所有的输入数据都进行检验，从而识别错误的输入，以保证每个数据的有效性；

§ 检查输入项的各种重要组合的合理性，必要时报告输入状态信息；

§ 使得输入的步骤和操作尽可能简单，并保持简单的输入格式；

§ 输入数据时，应允许使用自由格式输入；

§ 应允许缺省值；

§ 输入一批数据时，最好使用输入结束标志，而不要由用户指定输入数据数目；

§ 在以交互式输入／输出方式进行输入时，要在屏幕上使用提示符明确提示交互输入的请求，指明可使用选择项的种类和取值范围。同时，在数据输入的过程中和输入结束时，也要在屏幕上给出状态信息；

§ 当程序设计语言对输入／输出格式有严格要求时，应保持输入格式与输入语句的要求的一致性；

§ 给所有的输出加注解，并设计输出报表格式。

输入／输出风格还受到许多其它因素的影响。如输入／输出设备（例如终端的类型，图形设备，数字化转换设备等）、用户的熟练程度、以及通信环境等。

Wasserman 为“用户软件工程及交互系统的设计”提供了一组指导性原则，可供软件设计和编程参考。

§ 把计算机系统的内部特性隐蔽起来不让用户看到；

§ 有完备的输入出错检查和出错恢复措施，在程序执行过程中尽量排除由于用户的原因而造成程序出错的可能性；

§ 如果用户的请求有了结果，应随时通知用户；

§ 充分利用联机帮助手段，对于不熟练的用户，提供对话式服务，对于熟练的用户，提供较高级的系统服务，改善输入／输出的能力；

§ 使输入格式和操作要求与用户的技术水平相适应。对于不熟练的用户，充分利用菜单系统逐步引导用户操作；对于熟练的用户，允许绕过菜单，直接使用命令方式进行操作； § 按照输出设备的速度设计信息输出过程；

§ 区别不同类型的用户，分别进行设计和编码；

§ 保持始终如一的响应时间；

§ 在出现错误时应尽量减少用户的额外工作。

在交互式系统中，这些要求应成为软件需求的一部分，并通过设计和编码，在用户和系统之间建立良好的通信接口。

3. 程序效率

(1) 讨论效率的准则

程序的效率是指程序的执行速度及程序所需占用的内存的存储空间。讨论程序效率的几条准则为：

§ 效率是一个性能要求，应当在需求分析阶段给出。软件效率以需求为准，不应以人力所及为准。

§ 好的设计可以提高效率。

§ 程序的效率与程序的简单性相关。

一般说来，任何对效率无重要改善，且对程序的简单性、可读性和正确性不利的程序设计方法都是不可取的。

(2) 算法对效率的影响

源程序的效率与详细设计阶段确定的算法的效率直接有关。在详细设计翻译转换成源程序代码后，算法效率反映为程序的执行速度和存储容量的要求。

转换过程中的指导原则是：

§ 在编程序前，尽可能化简有关的算术表达式和逻辑表达式；

§ 仔细检查算法中的嵌套的循环，尽可能将某些语句或表达式移到循环外面；
§ 尽量避免使用多维数组；

§ 尽量避免使用指针和复杂的表；

§ 采用“快速”的算术运算；

§ 不要混淆数据类型，避免在表达式中出现类型混杂；

§ 尽量采用整数算术表达式和布尔表达式；

§ 选用等效的高效率算法；

许多编译程序具有“优化”功能，可以自动生成高效率的目标代码。它可剔除重复的表达式计算，采用循环求值法、快速的算术运算，以及采用一些能够提高目标代码运行效率的算法来提高效率。对于效率至上的应用来说，这样的编译程序是很有效的。

(3) 影响存储效率的因素

在大中型计算机系统中，存储限制不再是主要问题。在这种环境下，对内存采取基于操作系统的分页功能的虚拟存储管理，给软件提供了巨大的逻辑地址空间。这时，存储效率与操作系统的分页功能直接有关，并不是指要使所使用的存储空间达到最少。

采用结构化程序设计，将程序功能合理分块，使每个模块或一组密切相关模块的程序体积大小与每页的容量相匹配，可减少页面调度，减少内外存交换，提高存储效率。

在微型计算机系统中，存储容量对软件设计和编码的制约很大。因此要选择可生成较短目标代码且存储压缩性能优良的编译程序，有时需采用汇编程序。通过程序员富有创造性的努力，提高软件时间与空间效率。 提高存储效率的关键是程序的简单性。

(4) 影响输入／输出的因素

输入／输出可分为两种类型：一种是面向人（操作员）的输入／输出；一种是面向设备的输入／输出。如果操作员能够十分方便、简单地录入输入数据，或者能够十分直观、一目了然地了解输出信息，则可以说面向人的输入／输出是高效的。至于面向设备的输入／输出，分析起来比较复杂。从详细设计和程序编码的角度来说，可以提出一些提高输入／输出效率的指导原则：

§ 输入／输出的请求应当最小化；

§ 对于所有的输入／输出操作，安排适当的缓冲区，以减少频繁的信息交换。

§ 对辅助存储（例如磁盘），选择尽可能简单的，可接受的存取方法；

§ 对辅助存储的输入／输出，应当成块传送；

§ 对终端或打印机的输入／输出，应考虑设备特性， 尽可能改善输入／输出的质量和速度；

§ 任何不易理解的，对改善输入／输出效果关系不大的措施都是不可取的；

§ 任何不易理解的所谓“超高效”的输入／输出是毫无价值的；

§ 好的输入／输出程序设计风格对提高输入／输出效率会有明显的效果。

4. 程序设计语言

程序编码阶段的任务是将软件的详细设计转换成用程序设计语言实现的程序代码。因此，程序设计语言的性能和设计风格对于程序设计的效能和质量有着直接的关系。

(1) 程序设计语言特性的比较

① 软件心理学的观点

因为从设计到编码的转换基本上是人的活动，因此语言的性能对程序员的心理影响，将对转换产生重大影响。程序员总是希望选择简单易学、使用方便的语言，以减少程序出错率，提高软件可靠性。从心理学的观点，影响程序员心理的语言特性有如下六种：

§ 一致性 ：它表示一种语言所使用符号的兼容程度、允许随意规定限制、以及允许对语法或语义破例的程度。同是一个符号，给予多种用途，会引起许多难以察觉的错误。

§ 二义性 ：虽然语言的编译程序总是以一种机械的规则来解释语句，但读者则可能用不同的方式来理解语句。例如，对于一个逻辑表达式 A ≥“0”and A ≤“9”，读者可能对这个逻辑表达式有不同的理解。

如果一个程序设计语言缺乏一致性和存在二义性，那么用这种语言编写出来的程序可读性就差，同时用这种语言编程也容易出错。

§ 简洁性（紧凑性）：表示程序员为了用该语言编写程序，必须记忆的有关编码的信息量。可用语言支持块结构和结构化程序的能力、可使用的保留字和缩写字的种类、数据类型的种类和缺省说明、算术运算符和逻辑运算符的种类、系统内标准函数的数目等来衡量。

遗憾的是，语言的简洁性与程序的一致性常常是抵触的。

§ 局部性 ：指程序设计语言的联想（综合）特性。综合的特性使人们能够对事物从整体上进行记忆和识别。在编码过程中，由语句组合成模块，由模块组装为程序体系结构，并在组装过程中实现模块的高内聚和低耦合，可使程序的局部性加强。

§ 线性 ：指程序的联想（顺序）特性。人们总是习惯于按逻辑线性序列理解程序。如果程序中线性序列和逻辑运算较多，会提高可读性。如果存在大量的分支

和循环，就会破坏顺序状态，增加理解上的困难。直接实现结构化程序可提高程序的线性特性。

§ 传统 ：人们学习一种新的程序设计语言的能力受到传统的影响。具有 Pascal 基础的程序人员在学习 C 语言时不会感到困难，因为 C 保持了 Pascal 所确立的传统语言特性。但是要求同一个人去学习 APL 或者 LISP 这样一些语言，传统就中断了。

② 软件工程的观点

从软件工程观点，程序设计语言的特性应着重考虑软件开发项目的需要。为此，对于程序编码，有如下一些工程上的性能要求：

§ 详细设计应能直接地容易地翻译成代码程序 ：把设计变为程序的难易程度，反映了程序设计语言与设计说明相接近的程度。所选择的程序设计语言是否具有结构化的构造，复杂的数据结构，专门的输入 / 输出能力，位运算和串处理的能力，直接影响到从详细设计变换到代码程序的难易程度，以及特定软件开发项目的可实现性。

§ 源程序应具有可移植性 ：源程序的可移植性通常有三种解释：① 对源程序不做修改或少做修改就可以实现处理机上的移植或编译程序上的移植；② 即使程序的运行环境改变（例如，改用一个新版本的操作系统），源程序也不用改变；③ 源程序的许多模块可以不做修改或少做修改就能集成为功能性的各种软件包，以适应不同的需要。 为改善软件的可移植性，主要是使语言标准化。在开发软件时，应严格地遵守 ISO 或 ANSI、GB 的标准，而不要去理会特定编译器提供的非标准特性。

§ 编译程序应具有较高的效率。

§ 尽可能应用代码生成的自动工具 ：有效的软件开发工具是缩短编码时间，改善源代码质量的关键因素。使用带有各种有效的自动化工具的“软件开发环境”，支持从设计到源代码的翻译等各项工作，可以保证软件开发获得成功。

§ 可维护性 ：源程序的可读性，语言自身的文档化特性（涉及标识符的允许长度、标号命名、数据类型的丰富程度、控制结构的规定等）是影响到可维护性的重要因素。

③ 程序设计语言的技术性能

在计划阶段，极少考虑程序语言的技术特性。但在选定资源时，要规划将要使用的支撑工具，就要确定一个具体的编译器或者确定一个程序设计环境。如果软件开发组的成员对所要使用的语言不熟悉，那么在成本及进度估算时必须把学习的工作量估算在内。

一旦确定了软件需求，待选用的程序语言的技术特性就显得非常重要了。如果需要复杂的数据结构，就要仔细衡量有哪些语言能提供这些复杂的数据结构。如果首要的是高性能及实时处理的能力，就可选用适合于实时应用的语言或效率高的语言。如果该应用有许多输出报告或繁杂的文件处理，最好是根据软件的要求，选定一种适合于该项工作的语言。

软件的设计质量与程序设计语言的技术性能无关（面向对象设计例外）。但在实现软件设计转化为程序代码时，转化的质量往往受语言性能的影响。因而也会影响到设计方法。

语言的技术性能对测试和维护的影响是多种多样的。例如，直接提供结构化构造的语言有利于减少循环带来的复杂性（即 McCabe 复杂性），使程序易读、易测试、易维护。另一方面，语言的某些技术特性却会妨碍测试。例如，在面向对象的语言程序中，由于实行了数据封装，使得监控这些数据的执行状态变得比较困难；由于建立了对象类的继承结构，使得高内聚、低耦合的要求受到破坏，增加了测试的困难。此外，只要语言程序的可读性强，而且可以减少程序的复杂性，这样的程序设计语言对于软件的维护就是有利的。

总之，通过仔细地分析和比较，选择一种功能强而又适用的语言，对成功地实现从软件设计到编码的转换，提高软件的质量，改善软件的可测试性和可维护性是至关重要的。

(2) 程序设计语言的分类

目前，用于软件开发的程序设计语言已经有数百种之多，对这些程序设计语言的分类有不少争议。同一种语言可以归到不同的类中。从软件工程的角度，根据程序设计语言发展的历程，可以把它们大致分为 4 类。

① 从属于机器的语言（第一代语言）：它是由机器指令代码组成的语言。对于不同的机器就有相应的一套机器语言。用这种语言编写的程序，都是二进制代码的形式，且所有的地址分配都是以绝对地址的形式处理。存储空间的安排，寄存器、变址的使用都由程序员自己计划。因此使用机器语言编写的程序很不直观，在计算机内的运行效率很高但编写出的机器语言程序其出错率也高。

② 汇编语言（第二代语言）：汇编语言比机器语言直观，它的每一条符号指令与相应的机器指令有对应关系，同时又增加了一些诸如宏、符号地址等功能。存储空间的安排可由机器解决。不同指令集的处理器系统就有自己相应的汇编语言。从软件工程的角度来看，汇编语言只是在高级语言无法满足设计要求时，或者不具备支持某种特定功能（例如特殊的输入 / 输出）的技术性能时，才被使用。

③ 高级程序设计语言（第三代语言）

§ 传统的高级程序设计语言 ：如 FORTRAN、COBOL、ALGOL、BASIC 等。这些程序语言曾得到广泛应用。目前，它们都已有多种版本。有的语言得到较大的改进，

甚至形成了可视的开发环境，具有图形设计工具、结构化的事件驱动编程模式、开放的环境，使用户可以既快又简便地编制出 windows 下的各种应用程序。

§ 通用的结构化程序设计语言 ：它具有很强的过程功能和数据结构功能，并提供结构化的逻辑构造。这一类语言的代表是 PL/1，PASCAL，C 和 Ada。此外，COBOL78、Turbo BASIC 等也应归入第三代程序语言范围。

§ 专用语言 ：专用语言是为特殊的应用而设计的语言。通常具有自己特殊的语法形式，面对特定的问题，输入结构及词汇表与该问题的相应范围密切相关。有代表性的专用语言有 APL、Lisp、PROLOG、Smalltalk、C++、FORTH 等。从软件工程的角度来看，专用语言支持了特殊的应用，将特定的设计要求翻译成可执行的代码。但是它们的可移植性和可维护性比较差。

④ 第四代语言（4GL）：4GL 用不同的文法表示程序结构和数据结构，但是它是在更高一级抽象的层次上表示这些结构，它不再需要规定算法的细节。4GL 兼有过程性和非过程性的两重特性。程序员规定条件和相应的动作这是过程性的部分，并且指出想要的结果，这是非过程部分。然后由 4GL 语言系统运用它的专门领域的知识来填充过程细节。

Martin 把第四代语言分为以下几种类型：

§ 查询语言 ：用户可利用查询语言对预先定义在数据库中的信息进行较复杂的操作。

§ 程序生成器 ：只需很少的语句就能生成完整的第三代语言程序，它不必依赖预先定义的数据库作为它的着手点。

§ 其它 4GL ：如判定支持语言、原型语言、形式化规格说明语言等。

(3) 程序设计语言的选择

为某个特定开发项目选择程序设计语言时，既要从技术角度、工程角度、心理学角度评价和比较各种语言的适用程度，又必须考虑现实可能性。有时需要作出某种合理的折衷。

在选择与评价语言时，首先要从问题入手，确定它的要求是什么？这些要求的相对重要性如何？再根据这些要求和相对重要性来衡量能采用的语言。

通常考虑的因素有 ① 项目的应用范围；② 算法和计算复杂性；③ 软件执行的环境；④ 性能上的考虑与实现的条件；⑤ 数据结构的复杂性；⑥ 软件开发人员的知识水平和心理因素等。其中，项目的应用范围是最关键的因素。

针对计算机的 4 个主要应用领域，为语言做一个粗略的分类。例如，在科学与工程计算领域内，C，C++ 语言得到了广泛的应用，但 FORTRAN 仍然是应用最广泛的语言。在商业数据处理领域中，通常采用 COBOL，RPG 语言编写程序，当然也

可选用 SQL 语言或其它专用语言。在系统程序设计和实时应用领域中，汇编语言或一些新的派生语言，如 BLISS，PL / S，Ada，C++等得到了广泛的应用。在人工智能领域以及问题求解，组合应用领域，主要采用 LISP 和 PROLOG 语言。

新的更强有力的语言，虽然对于应用有很强的吸引力，但是因为已有的语言已经积累了大量的久经使用的程序，具有完整的资料、支撑软件和软件开发工具，程序设计人员比较熟悉，而且有过类似项目的开发经验和成功的先例，由于心理因素，人们往往宁愿选用原有的语种。所以应当彻底地分析，评价，介绍新的语言，以便从原有语言过渡到新的语言。

5. 程序复杂性度量

程序复杂性主要指模块内程序的复杂性。它直接关联到软件开发费用的多少，开发周期的长短和软件内部潜伏错误的多少。同时它也是软件可理解性的另一种度量。

减少程序复杂性，可提高软件的简单性和可理解性，并使软件开发费用减少，开发周期缩短，软件内部潜藏错误减少。

(1) 代码行度量法

度量程序的复杂性，最简单的方法就是统计程序的源代码行数。此方法基于两个前提：

• 程序复杂性随着程序规模的增加不均衡地增长；

• 控制程序规模的方法最好是采用分而治之的办法。将一个大程序分解成若干个简单的可理解的程序段。

方法的基本考虑是统计一个程序模块的源代码行数目，并以源代码行数做为程序复杂性的度量。若设每行代码的出错率为每 100 行源程序中可能有的错误数目，例如每行代码的出错率为 1%，则是指每 100 行源程序中可能有一个错误。

Thayer 曾指出，程序出错率的估算范围是从 0.04%～7%之间，即每 100 行源程序中可能存在 0.04～7 个错误。他还指出，每行代码的出错率与源程序行数之间不存在简单的线性关系。Lipow 进一步指出，对于小程序，每行代码的出错率为 1.3%～1.8%； 对于大程序，每行代码的出错率增加到 2.7%～3.2%之间，但这只是考虑了程序的可执行部分，没有包括程序中的说明部分。Lipow 及其他研究者得出一个结论：对于少于 100 个语句的小程序，源代码行数与出错率是线性相关的。随着程序的增大，出错率以非线性方式增长。所以，代码行度量法只是一个简单的，估计得很粗糙的方法。

(2) McCabe 度量法

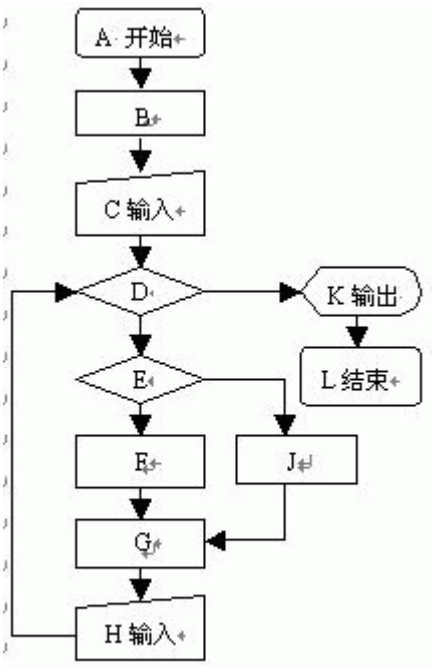


图 4.3 程序图的例子

McCabe 度量法是一种基于程序控制流的复杂性度量方法。McCabe 定义的程序复杂性度量值又称环路复杂度，它基于一个程序模块的程序图中环路的个数。
如果把程序流程图中每个处理符号都退化成一个结点，原来联结不同处理符号的流线变成连接不同结点的有向弧，这样得到的有向图就叫做程序图。
计算有向图 G 的环路复杂性的公式：V (G)＝m－n＋2
其中，V（G）是有向图 G 中的环路个数，m 是图 G 中有向弧个数，n 是图 G 中结点个数。

以图 4.3 为例，其中，结点数 n＝11，弧数 m＝12，则有 V（G）＝m－n＋2＝12－11＋2＝3。即 McCabe 环路复杂度度量值为 3。 它也可以看做由程序图中的有向弧所封闭的区域个数。

当分支或循环的数目增加时，程序中的环路也随之增加，因此 McCabe 环路复杂度度量值实际上是为软件测试的难易程度提供了一个定量度量的方法，同时也间接地表示了软件的可靠性。实验表明，源程序中存在的错误数以及为了诊断和纠正这些错误所需的时间与 McCabe 环路复杂度度量值有明显的关系。

Myers 建议，对于复合判定，例如（A＝0）∩（C＝D）∪（X＝'A'）算做三个判定。

利用 McCabe 环路复杂度度量时，有几点说明。

§ 环路复杂度取决于程序控制结构的复杂度。当程序的分支数目或循环数目增加时其复杂度也增加。环路复杂度与程序中覆盖的路径条数有关。

§ 环路复杂度是可加的。例如，模块 A 的复杂度为 3，模块 B 的复杂度为 4，则模块 A 与模块 B 的复杂度是 7。

§ McCabe 建议，对于复杂度超过 10 的程序，应分成几个小程序，以减少程序中的错误。Walsh 用实例证实了这个建议的正确性。他发现，在 McCabe 复杂度为 10 的附近，存在出错率的间断跃变。

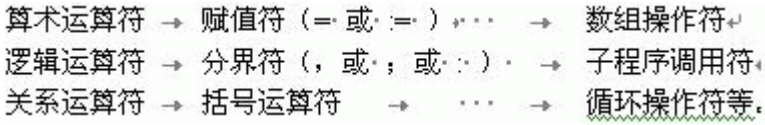
§ McCabe 环路复杂度隐含的前提是：错误与程序的判定加上例行子程序的调用数目成正比。而加工复杂性、数据结构、录入与打乱输入卡片的错误可以忽略不计。

(3) Halstead 的软件科学

Halstead 软件科学研究确定计算机软件开发中的一些定量规律，它采用以下一组基本的度量值，这些度量值通常在程序产生之后得出，或者在设计完成之后估算出。

① 程序长度，即预测的 Halstead 长度

令 n1 表示程序中不同运算符（包括保留字）的个数，令 n2 表示程序中不同运算对象的个数，令 H 表示“程序长度”，则有 H = n1 * log2 n1 + n2 * log2 n2 这里，H 是程序长度的预测值，它不等于程序中语句个数。在定义中，运算符包括：



特别地，成对的运算符，例如“BEGIN…END”、“FOR…TO”、“REPEAT …UNTIL”、“WHILE…DO”、“IF…THEN…ELSE”、“（…）”等都当做单一运算符。

运算对象包括变量名和常数。

② 实际的 Halstead 长度

设 N1 为程序中实际出现的运算符总个数，N2 为程序中实际出现的运算对象总个数， N 为实际的 Halstead 长度，则有 N = N1 + N2

③ 程序的词汇表

Halstead 定义程序的词汇表为不同的运算符种类数和不同的运算对象种类数的总和。若令 n 为程序的词汇表，则有 n = n1 + n2

SUBROUTINE SORT（X，

运算符	计数	运算对象	计数
可执行语句结束	7	X	6
数组下标	6	I	5
=	5	J	4
IF（ ）	2	N	2
DO	2	2	2
,	2	SAVE	2
程序结束	1	1	1
LT	1	n2=7 N2=22	
GE	1		
GO TO 10	1		
n1=10 N1=28			

N）

```
DIMENSION X（N）
IF（N .LT. 2） RETURN
DO 20 I = 2, N
DO 10 J = 1, I
IF （X（I）.GE. X（J））

GO TO 10
SAVE = X（I）
X（I）= X（J）
X（J）= SAVE
10 CONTINUE
20 CONTINUE
RETURN
END
```

因此有： 预测的词汇量 H = n1* log2 n1 + n2* log2 n2 = 10 * log2 10 + 7 * log2 7 = 52.87

实际的词汇量 N = N1 + N2 = 28 + 22 = 50

程序的词汇表 n = n1 + n2 = 10 + 7 = 17

④ 程序量 V，可用下式算得
V = （ N1 + N2 ） * log2 （ n1 + n2 ）

它表明了程序在“词汇上的复杂性”。其最小值为

V* = （ 2 + n2*） * log2 （ 2 + n2* ）

这里，2 表明程序中至少有两个运算符：赋值符“:=”和函数调用符“f（）”，n2* 表示输入／输出变量个数。对于图 4.4 的例子，利用 n1，N1，n2，N2，可以计算得

V = （ 28 + 22 ）* log2 （ 10 + 7 ）= 204

等效的汇编语言程序的 V＝328。这说明汇编语言比 FORTRAN 语言需要更多的信息量（以 bit 表示）。

⑤ 程序量比率（语言的抽象级别）

L = V*／V 或 L = （ 2／n1 ）*（ n2／N2 ）
这里，N2 = n2 * log2 n2。它表明了一个程序的最紧凑形式的程序量与实际程序量之比，反映了程序的效率。其倒数

D = 1／L

表明了实现算法的困难程度。有时，用 L 表达语言的抽象级别，即用 L 衡量在表达程序过程时的抽象程度。对于高级语言，它接近于 1，对于低级语言，它在 0～1 之间。下面列出的是根据经验得出的一些常用语言的语言抽象级别。

语　言	L 的平均值
English Prose（英语散文）	2.16
PL/1	1.53
ALGOL68	2.12
FORTRAN	1.14
Assembler（汇编语言）	0.88

⑥ 程序员工作量

E = V／L

⑦ 程序的潜在错误

Halstead 度量可以用来预测程序中的错误。 认为程序中可能存在的差错应与程序的容量成正比。因而预测公式为

B = （ N1 + N2 ）* log2 （ n1 + n2 ）／3000 = V／3000
B 表示该程序的错误数。例如，一个程序对 75 个数据库项共访问 1300 次，对 150 个运算符共使用了 1200 次，那么预测该程序的错误数：

B = （ 1300 + 1200 ）* log2 （ 75 + 150 ）／3000 = 6.5
即预测该程序中可能包含 6～7 个错误。

⑧ Halstead 的重要结论之一是：程序的实际 Halstead 长度 N 可以由词汇表 n 算出。即使程序还未编制完成，也能预先算出程序的实际 Halstead 长度 N， 虽然它没有明确指出程序中到底有多少个语句。

这个结论非常有用。经过多次验证，预测的 Halstead 长度与实际的 Halstead 长度是非常接近的。

Halstead 度量是目前最好的度量方法。 但它也有缺点：

§ 没有区别自己编的程序与别人编的程序。这是与实际经验相违背的。这时应将外部调用乘上一个大于 1 的的常数 Kf（应在 1～5 之间，它与文档资料的清晰度有关）。

§ 没有考虑非执行语句。补救办法：在统计 n1、n2、N1、N2 时，可以把非执行语句中出现的运算对象，运算符统计在内。

§ 在允许混合运算的语言中，每种运算符必须与它的运算对象相关。如果一种语言有整型、实型、双精度型三种不同类型的运算对象，则任何一种基本算术运算符（＋、－、×、／）实际上代表了 $A_3^2 = 6$ 种运算 符。如果语言中有 4 种不同类型的算术运算对象， 那么每一种基本算术运算符实际上代表了 $A_4^2 = 12$ 种运算符。 在计算时应考虑这种因数据类型而引起差异的情况。

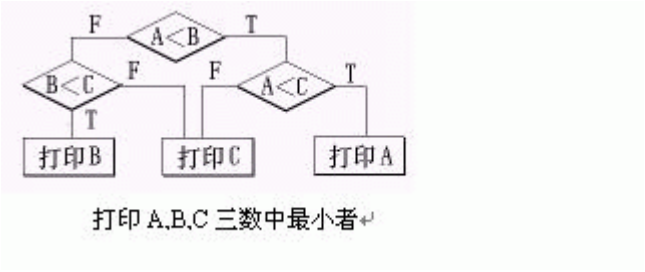
§ 没有注意调用的深度。Halstead 公式应当对调用子程序的不同深度区别对待。在计算嵌套调用的运算符和运算对象时，应乘上一个调用深度因子。这样可以增大嵌套调用时的错误预测率。

§ 没有把不同类型的运算对象，运算符与不同的错误发生率联系起来，而是把它们同等看待。例如，对简单 IF 语句与 WHILE 语句就没有区别。实际上，WHILE 语句复杂得多，错误发生率也相应地高一些。

§ 忽视了嵌套结构（嵌套的循环语句、嵌套 IF 语句、括号结构等）。一般地，运算符的嵌套序列，总比具有相同数量的运算符和运算对象的非嵌套序列要复杂得多。解决的办法是对嵌套结果乘上一个嵌套因子。

三、例题分析

【例 1】下图是使用 Basic 语言编写的一个打印 A，B，C 三数中最小者的程序的流程图。其中出现了 6 个 GOTO 语句，一个向前，5 个向后，程序可读性很差。



打印 A,B,C 三数中最小者

```
if ( A < B ) goto 120;
if ( B < C ) goto 110;
100 print C;
goto 140;
110 print B;
goto 140;
120 if ( A < C ) goto 130;
goto 100;
130 print A;

140
```

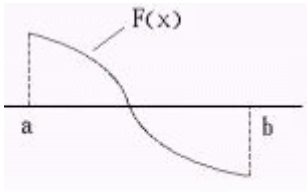
试利用基本控制结构，将程序中的 GOTO 语句消去。

答案：使用 if - then - else 结构化构造，则上述程序段可改成如下形式。

```
if ( A < B and A < C ) then
print A
else if ( A >= B and B < C ) then
print B
else
print C;
```

分析:理清程序流程图中每一条执行路径,适当利用复合的条件测试于判断语句，对每一个最终的打印处理，保留一个分支进入它。这样可以消除众多的 GOTO 语句，甚至可以消除嵌套的判断语句结构。这种程序结构清晰，可读性好。

【例 2】设在闭区间 [a..b] 上函数 F(X) 有唯一的一个零点，如下图所示。下面给出一个用 C 语言写出的程序段,用二分法求方程 F(X)＝0 在区间 [a..b] 中的根。程序段中 X0、X1 是当前求根区间 [X0..X1] 的下上界，Xm 是该区间的中点，eps 是一个给定的很小正数，用于迭代收敛的判断。在程序中采取了用 goto 语句和标号 finish 控制在循环中途转出循环。



函数F(X)曲线

```
〖程序〗 F0 = F (a); F1 = F (b);
if ( F0 * F1 <= 0 ) {

X0 = a; X1 = b;
for ( i = 1; i <= n; i++) {

Xm = (X0 + X1) / 2; Fm = F(Xm);
if ( abs (Fm) < eps || abs (X1－X0) < eps )
goto finish;
if ( F0 * Fm > 0 )
{ X0 = Xm; F0 = Fm; }
else
X1 = X;
}
finish： printf (“\n The root of this equation is %d\n”, Xm );
}
```

这类循环结构出现了两个循环出口。一个是 for 循环的正常出口：当循环控制变量 i 超出了循环终值 n 时退出循环；另一个是 for 循环的非正常出口：当某种条件满足时，从循环中间某处转出循环，执行循环后面的语句。它不满足结构化的要求。

试利用结构化程序设计要求的几种基本控制结构，消除其中的 goto 语句，使得每一个部分都是单入口单出口。
答案：利用一个布尔变量 finished（该变量的初值为 false），当循环中求到了要求的结果时，将此变量的值改变为 true，表示循环应结束，while 循环测试到 finished 为 true，就自动退出循环，执行后续的语句。

```
〖程序〗 F0 = F(a); F1 = F(b);
if ( F0 * F1 <= 0 ) {

X0 = a; X1 = b; i = 1; finished = 0; /* 设置控制循环结束的布尔变量 */

while ( i <= n && finished == 0 ) { /* 单入口单出口 */

Xm = (X0 + X1) / 2; Fm = F(Xm);
if ( abs(Fm) < eps | | abs(X1－X0) < eps ) finished = 1;

if ( finished == 0) {
```

```
if ( F0 * Fm > 0 )
{ X0 = Xm; F0 = Fm; }
else
X1 = X;

}
}
}
```

分析：此程序段中各种结构均为单入口与单出口，且没有 goto 语句，可以说它是一个结构化的程序。它的结构很简单，只有一重循环，但由于引入一个布尔变量来控制循环结束，可读性比较差。在只有一重循环的情形，差的程度还不很明显，在多重循环的情形，引入多个布尔变量，可读性就很差了。理解程序的时间差几倍到几十倍不止。因此，对于这种单入口多出口的循环，R.S.Pressman 明确说明，用 goto 语句可得到较好的清晰性。在 C 中提供了一个可直接转向循环之后的语句 break，使用它可获得较好的效果。

```
〖程序〗 F0 = F(a); F1 = F(b); /*区间两端点的函数值*/
if ( F0 * F1 <= 0 ) { /*区间中没有根，不做*/
X0 = a; X1 = b; /*设置当前求根区间的两个端点*/
for ( i = 1; i <= n; i++) { /*最多允许迭代 n 次*/
Xm = (X0 + X1) / 2; Fm = F(Xm); /*求中点及中点的函数值*/
if ( abs(Fm) < eps || abs(X1－X0) < eps) break; /*求到, 转出循环*/
if ( F0 * Fm > 0 ) /*没有求到，缩小求根区间*/
{ X0 = Xm; F0 = Fm; } /*向右缩小区间*/
else
X1 = X; /*向左缩小区间*/
}
}
```

这段程序不是结构化的程序，利用了 C 语言中的一个语句 break，它的功能是将控制转移到它所在循环的后面第一个后续语句处。由于将转移语句与转出条件的判断直接联系在一起，可读性较好。

【例 3】试说明下面的两个程序段的功能是什么？可否用另一些等效的程序段来代替它，以提高其可读性。

```
(1) A[I] = A[I] + A[T];
A[T] = A[I] - A[T];
A[I] = A[I] - A[T];
```

```
(2) for ( i = 1; i <= n; i ++ )
for ( j = 1; j <= n; j ++ )
V[i][j] = ( i / j ) * ( j / i );
```

答案：(1) 的功能是对换 A[I] 与 A[T] 的内容。等效的程序段可以是：

```
WORK = A[T]; A[T] = A[I]; A[I] = WORK;
```

(2) 的功能是建立一个单位矩阵 V。等效的程序段可以是：

```
for ( i = 1; i <= n; i ++ )

for ( j = 1; j <= n; j ++ )

if ( i == j ) V[i][j] = 1;

else V[i][j] = 0;
```

分析：阅读这两段程序，读者可能不易看懂，有时还需用实际数据试验一下。对于(1)，如果我们给 A[I] 赋值 3，给 A[T] 赋值 5，在运算后发现 A[I] 中变成了 5，A[T] 中变成了 3。这段程序就是交换 A[I] 和 A[T] 中的内容。目的是为了节省一个工作单元。如果改一下：

```
WORK = A[T]; A[T] = A[I]; A[I] = WORK;
```

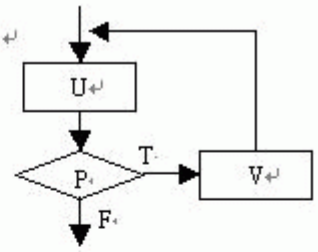
就能让读者一目了然。

对于(2)，除法运算（ / ）在除数和被除数都是整型量时，其结果只取整数部分，而得到整型量。因此， i / j 为 0， 当 i < j 时
j / i 为 0， 当 j < i 时
得到的数组 V[i][j] = (i / j) * (j / i) = 0，当 i ≠ j 时
(i / j) * (j / i) = 1，当 i ＝ j 时
这样得到的结果 V 是一个单位矩阵。如果直截了当地说明作者的意图：

```
if ( i == j ) V[i][j] = 1; else V[i][j] = 0;
```

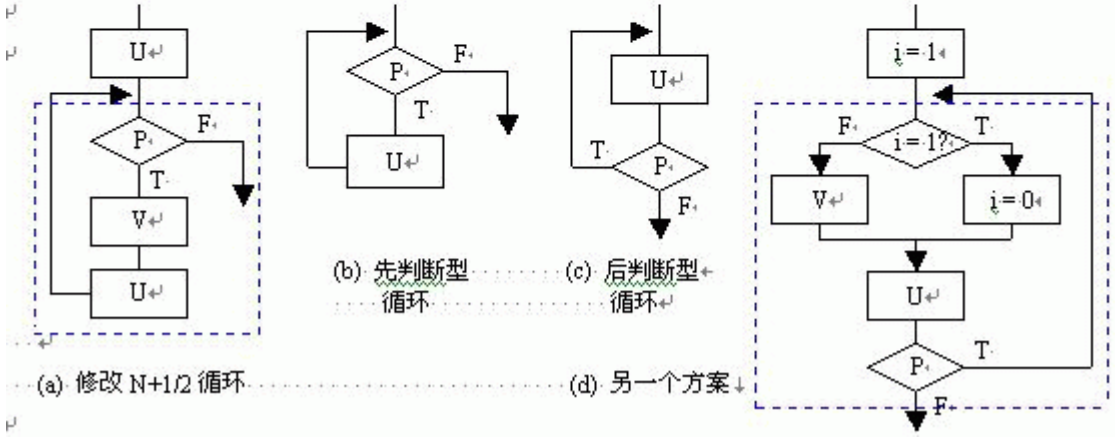
让读者可以很容易地理解。所以，在程序设计时，应当首先考虑清晰性，不要玩技巧。

【例 4】有一种循环结构，叫做 N+1/2 循环。其流程图如下所示。这种控制结构不属于基本控制结构：它既不是先判断型循环，又不是后判断型循环。试修改此流程图，将它改为用基本控制结构表示的等效的流程图。

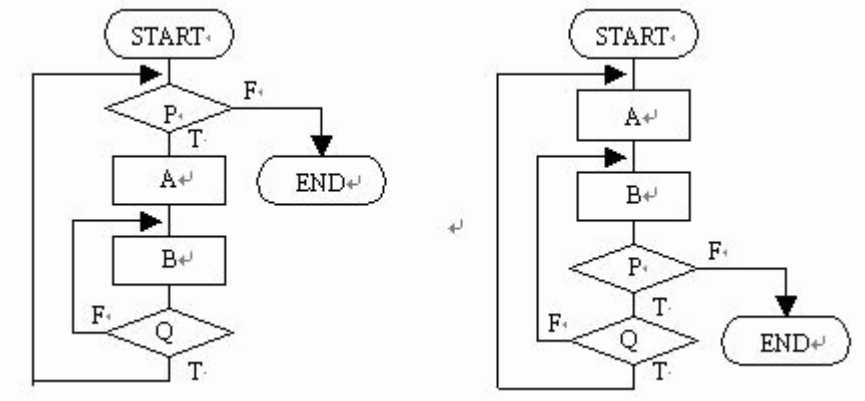


答案：等效的控制流程图如下图中(a)所示。

分析：先判断型循环要求在进入循环体之前，先判断是否要继续执行此循环。因此，在这种控制结构的入口处应是一个判断语句。这种循环的循环体可能一次也不执行。参看图(b)。

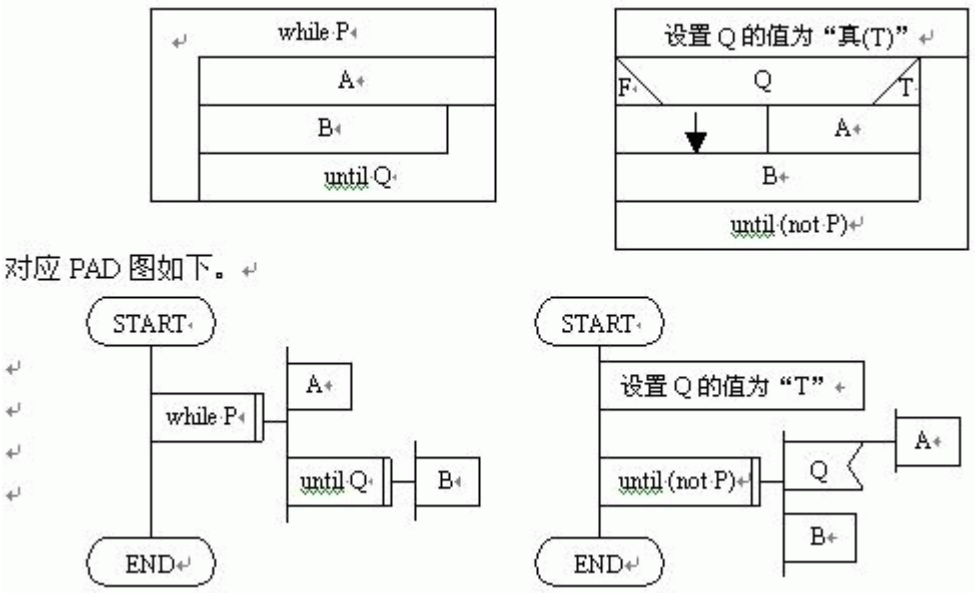


【例 5】下面是两个程序流程图，试分别用 N-S 图和 PAD 表示之，并计算它们的 McCabe 复杂性度量。



答案：对应的 N-S 图如下。

整理人：刘海云
整理时间：2003.05



McCabe 复杂性度量都为 3。

四、习题

【4-1】从下列关于模块化程序设计的叙述中选出 5 条正确的叙述。

① 程序设计比较方便，但比较难以维护。

② 便于由多个人分工编制大型程序。

③ 软件的功能便于扩充。

④ 程序易于理解，也便于排错。

⑤ 在主存储器能够容纳得下的前提下，应使模块尽可能大，以便减少模块的个数。

⑥ 模块之间的接口叫做数据文件。

⑦ 只要模块之间的接口关系不变，各模块内部实现细节的修改将不会影响别的模块。

⑧ 模块间的单向调用关系叫做模块的层次结构。

⑨ 模块越小，模块化的优点越明显。一般来说，模块的大小都在 10 行以下。

【4-2】结构化程序设计有时被错误地称为“无 GOTO 语句”的程序设计。请说明为什么会出现这样的说法，并讨论环绕着这个问题的一些争论。

【4-3】从下面关于程序编制的叙述中，选出三条正确的叙述。

① 在编制程序之前，首先必须仔细阅读给定的程序说明书。然后，必须如实地依照说明书编写程序。说明书中常会有含糊不清或难以理解的地方。程序员在作业时应该对这些地方作出适当的解释。

② 在着手编制程序时，重要的是采用既能使程序正确地按设计说明书进行处理，又易于出错的编写方法。

③ 在编制程序时，首先应该对程序的结构充分考虑，不要急于开始编码，而要象写软件文档那样，很好地琢磨程序具有什么样的功能，这些功能如何安排等等。

④ 考虑到以后的程序变更，为程序编写完整的说明书是一项很重要的工作。只要有了完整的程序说明书，即使程序的编写形式难以让他人看懂也没有什么关系。

⑤ 编制程序时不可缺少的条件是，程序的输入和输出数据的格式都应确定。其他各项规定都是附带的，无足轻重。

⑥ 作为一个好的程序，不仅处理速度要快，而且易读易修改等等也都是重要的条件。为了能得到这样的程序，不仅要熟悉程序设计语言的语法，还要注意采用适当的规程和单纯的表现方法，注意使整个程序的结构简洁。

【4-4】从下列叙述中选出 5 条符合程序设计风格指导原则的叙述。

① 嵌套的重数应加以限制。

② 尽量多使用临时变量。

③ 不滥用语言特色。

④ 不用可以省略的括号。

⑤ 使用有意义的变量名。

⑥ 应尽可能把程序编得短些。

⑦ 把常见的局部优化工作留给编译程序去做。

⑧ 注解越少越好。

⑨ 程序的格式应有助于读者理解程序。

⑩ 应尽可能多用 GOTO 语句。

【4-5】从供选择的答案中选出应该填入下面（ ）中的正确答案。

A. 允许用户建立、修改、存储正文的计算机程序是（ ）。

① BOOtstrap ② Editor ③ Loader ④ Textformatter

B. 程序语言的编译系统和解释系统相比，从用户程序的运行效率来看（ ）。

① 前者运行效率高 ② 两者大致相同

③ 后者运行效率高 ④ 不能确定

C. FORTRAN 语言的源程序是（ ）结构。

① 块状 ② 分程序嵌套

③ 既是块状，又是嵌套 ④ 既不是块状，又不是嵌套的

D. 国际上最广泛使用的商用及行政管理语言是（ ）。

① COBOL ② BASIC ③ FORTRAN ④ PL/1

E. 国际上最流行的数值计算的程序设计语言是（ ）。

① BASIC ② ALGOL ③ FORTRAN ④ C

F. 美国国防部主持开发了高级程序设计语言 Ada，在它研制开始时，经反复比较，确定以高级语言（ ）作为 Ada 研究的出发点。

① LISP ② ALGOL ③ ALGOL68 ④ PL/1

G. 在人工智能领域，目前最广泛使用的高级语言是（ ）。

① Ada ② FORTRAN ③ COBOL ④ LISP

【4-6】从供选择的答案中选出应该填入下面（ ）中的正确答案。

A. 汇编程序是指（ ）。

① 用汇编语言写的程序 ② 符号程序 ③ 汇编语言的处理程序

B. 为了实现递归子程序的正确调用，人们必须用（ ）来保存（ ）及有关信息。

① 堆栈 ② 线性表 ③ 队列 ④ 树

⑤ 入口点 ⑥ 返回地址 ⑦ 断点

C. UNIX 操作系统是（ ）研制的，它是用程序语言（ ）书写实现的。

① Bell 实验室 ② DEC 公司 ③ IBM 公司 ④ PASCAL

⑤ 并发 PASCAL ⑥ MODULA ⑦ C

【4-7】下面给出一个求实函数方程 F(x)在自变量区间 [a, b] 中的全部实根的算法。首先阅读此程序，然后

(1) 画出消去全部 goto 语句的结构化程序流程图。

(2) 将它改成 N_S 图。

(3) 计算该程序的 McCabe 复杂性度量。

在算法中，a 与 b 是区间[a, b]的两端点值；eps1 与 eps2 是用户要求的求解精度。如果区间中点的函数值的绝对值小于 eps1 或新的小区间的长度小于 eps2，就认为这个中点为根。

```
float BinRoot ( float a, float b, float eps1, float eps2 ) {

float low= a, high = b, mid, fmid;

float flow = Func(low), fhigh := Func(high);

label L1, L2, L3; //标号说明，给定某些程序地址

if ( flow * fhigh > 0.0 ) { BinRoot = 0; goto L3; } //无实根

L1: mid = (low + high) / 2; fmid = Func(mid);

if ( abs ( fmid ) <= eps1 ) {

L2: BinRoot = mid; goto L3;

}

else if ( high - mid <= eps2 ) goto L2;

else if ( flow * fmid > 0.0 ) { low = mid; flow = fmid; goto L1; }

else { high = mid; goto L1 };

L3:

}
```

【4-8】从供选择的答案中选出适当的字句填入下面关于程序生产率的描述中的( ) 内。

(1) 1960 年底 Dijkstra 提倡的 （ A ） 是一种有效的提高程序设计效率的方法。

(2) Dijkstra 为了使程序结构易于理解，把基本控制结构限于顺序、（B）、（C）3 种，应避免使用 （ D ）。

(3) （ A ） 不仅提高程序设计的生产率，同时也容易进行程序的 （ E ）。

供选择的答案：

A. ① 标准化程序设计 ② 模块化程序设计 ③ 多道程序设计

④ 宏语言 ⑤ 结构化程序设计 ⑥ 汇编语言

⑦ 表格处理语言

B, C. ① 分支 ② 选择 ③ 重复 ④ 计算 ⑤ 输入输出

D. ① GOTO 语句 ② DO 语句 ③ IF 语句 ④ REPEAT 语句

E. ① 设计 ② 调试 ③ 维护 ④ 编码

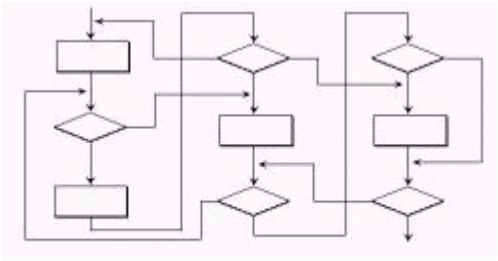
【4-9】用某种软件复杂性度量算法来度量不同类型的程序时，得出的度量值是否真正反映了它们的复杂性？如果对同类型的程序进行度量，其结果是否就比较有价值？

【4-10】软件复杂性有哪几类？软件复杂性度量模型应遵循哪些基本原则？

五、习题解答

【4-1】正确的叙述有②、③、④、⑦、⑧。如果程序结构的模块化满足评价的标准（高内聚，低耦合），这样的结构是容易编码，容易测试，容易理解，容易修改，容易维护的。程序的功能也容易扩充。特别适合于大型程序编制时，多人分工合作，协同完成任务的情形。因为是采用自顶向下，逐层分解来划分模块结构的，所以模块之间的调用关系是分层次的模块结构，就叫做模块的层次结构。模块之间的信息传递叫做模块的接口，模块之间传递信息可以通过参数表、全局变量或全局数据结构、数据文件、专门的通信模块，不是专指数据文件。划分模块时，模块大小要适中。模块太大，控制路径数目多、涉及的范围广、变量的数目多、总体复杂性高，可理解性、可修改性、可靠性就会变差。模块太小，模块个数增大，调用的系统开销就会增大。所以要有一个权衡。

【4-2】早在 1963 年，针对当时流行的 ALGOL 语言，Peter Naur 指出，在程序中大量地，没有节制地使用 GOTO 语句，会使程序结构变得非常混乱。但是很多人还不太注意这一问题。以致许多人写出来的程序仍然是纷乱如麻的。



结构化的程序流程

1965 年,E.W.Dijkstra 在一次会议上提出,应当把 GOTO 语句从高级语言中取消。并指出，程序的质量与程序中包含的 GOTO 语句的数量成反比。在这种思想的影响下，当时新开发的几种高级程序设计语言，例如 LISP、ISWIM、BLISS 等，都把 GOTO 语句取消了。

1966 年,Bohm 与 Jacopini 证明了任何单入口单出口的没有“死循环”的程序都能由三种最基本的控制结构构造出来。这三种基本控制结构就是“顺序结构”、“选择 IF－THEN－ELSE 结构”、“重复 DO－WHILE 或 DO－UNTIL 结构”。

1968 年，Dijkstra 在写给<ACM>（美国计算机协会通讯）杂志编辑部的信中再次建议从一切高级语言中取消 GOTO 语句，只使用三种基本控制结构编写程序。他的建议引起了激烈的争论。争论集中在如何看待 GOTO 语句的问题上。赞成取消 GOTO 语句的一方认为，GOTO 语句对程序清晰性有很大破坏作用，凡是使用 GOTO 语句多的程序，其控制流时而 GOTO 向前，时而 GOTO 向后，常使程序变得很难理解，从而增加查错和维护的困难，降低程序的可维护性。但以 D.E.Knuth 为代表的另一方认为，GOTO 语句虽然存在着破坏程序清晰性的问题，但不应完全禁止。因为 GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁，并且 GOTO 语句可直接得到硬件指令的支持。经过争论，人们认识到，不是简单地去掉 GOTO 语句的问题，而是要创立一种新的程序设计思想、方法和风格，以显著提高软件生产率和软件质量，降低软件维护的成本。

70 年代初 N.Wirth 在设计 Pascal 语言时对 GOTO 语句的处理可被当做对 GOTO 语句争论的结论。在 Pascal 语言中设置了支持上述三种基本控制结构的语句；另一方面，GOTO 语句仍然保留在该语言中。不过，N.Wirth 解释说，通常使用所提供的几种基本控制结构已经足够,习惯于这样做的人不会感到 GOTO 语句的必要。也就是说，在一般情况下，可以完全不使用 GOTO 语句。如果在特殊情况下，由于特定的要求，偶然使用 GOTO 语句能解决问题，那也未尝不可，只是不应大量使用罢了。

事实上，大量采用 GOTO 语句实现控制路径，会使程序路径变得复杂而且混乱，从而使程序变得不易阅读,给程序的测试和维护造成困难,还会增加出错的机会，降低程序的可靠性。因此要控制 GOTO 语句的使用。但有时完全不用 GOTO 语句进行程序编码，比用 GOTO 语句编出的程序可读性差。例如，在查找结束时，文件访问结束时，出现错误情况要从循环中转出时，使用布尔变量和条件结构来实现就不如用 GOTO 语句来得简洁易懂。

【4-3】①、④、⑥。编制程序的过程实际上是根据设计的结果，用某种机器能够识别的程序设计语言，将设计翻译成机器代码的过程。因此，必须如实地按照设计说明书编写程序。至于设计说明书中含糊不清的地方，应当在编程时与分析人员或设计人员协商，对这些地方做出适当的解释。另外，考虑到将来的程序修改，必须为程序编写完整的说明书，同时程序必须编写得容易让别人看得懂，这样程序才容易修改，修改时不容易出错，而且容易验证修改后得结果。还有，编写程序的人不须重新考虑程序要完成什么功能，这些已经在软件分析与设计过程中充分考虑过了。

【4-4】①、③、⑤、⑦、⑨是正确的。因为 ① 条件语句和循环语句嵌套得过多会增加程序的复杂性，从而增加程序的出错率。③ 虽然国际以至国内已经发表了编程语言的标准，但各个计算机厂商在推出自己的计算机系统的同时，也推出了针对自己机器特色的程序设计语言的非标准版本，如果利用这些语言的非标准特性编写程序，就会给将来程序的移植带来困难。为了提高程序的可移植性，应当只使用语言的标准版本，不要滥用语言的非标准特色。⑤ 给在程序中使用的变量赋予与实际含义相符的名字，可以提高程序的可读性，从而提高程序的可维护性。⑦ 程序优化的工作最好交给编译程序来做，程序员应把主要注意力放在提高程序的可读性、清晰性、简洁性、正确性、一致性等方面，从而保证软件的可靠性和可维护性。⑨ 程序的可读性是至关重要的。所以程序的格式应有助于读者理解程序。

【4-5】A. ②, B. ①, C. ①, D. ①, E. ③, F. ②, G. ④

除以上几种论述外，其它的叙述都不对。例如，程序中加入临时变量，可能会改变程序执行中的时序关系，造成程序出错。在表达式中加入括号，可以明确标明表达式的运算优先关系，避免因语言方面的原因可能潜藏的错误。程序模块的大小要适中，不是编得越短越好。注解加多少，由问题得难度来决定，但决不是可有可无的。最后要限制 GOTO 语句的使用，因为它可能会造成思路混乱、极易出错。

A.计算机用户通常是使用“编辑程序（Editor）”对源程序文本进行录入、编辑、存储的，不用自举程序（Bootstrap）、连接程序（Loader）或文本格式化程序（Textformatter）。

B. 解释系统是边解释源程序边执行该源程序，编译程序是先编译出源程序的对应目标代码，再执行这些目标代码。所以编译程序编出的目标代码运行效率高。

C. FORTRAN 程序是以 SUBROUTINE 为单元的块状结构，对每一个 SUBROUTINE 进行编译后通过连接形成整个程序系统。它不是嵌套的。

D. 国际上最流行的商业和行政管理语言是 COBOL 语言。

E. 国际上最流行的用于数值计算的语言是 FORTRAN 语言。

F. 美国国防部主持开发高级程序设计语言 Ada 时，曾确定以 ALGOL 语言作为 Ada 研究的出发点。所以，Ada、ALGOL、Pascal、BASIC 和 C 都是 ALGOL 系的一些程序语言。

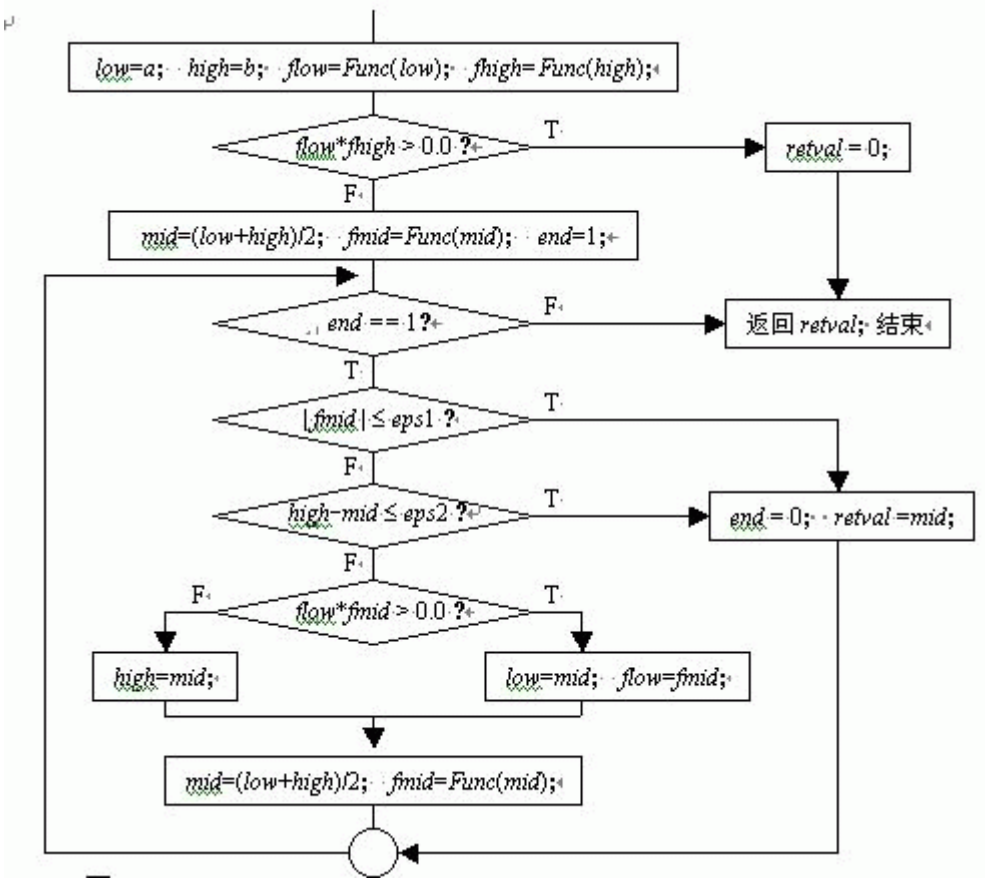
G. 在人工智能领域，目前最广泛使用的高级语言是 Lisp。

【4-6】A. ③ B. ①、⑥ C. ①、⑦ A. 汇编程序实际是指汇编语言的处理程序。而用汇编语言写成的源程序一般称为汇编语言程序。

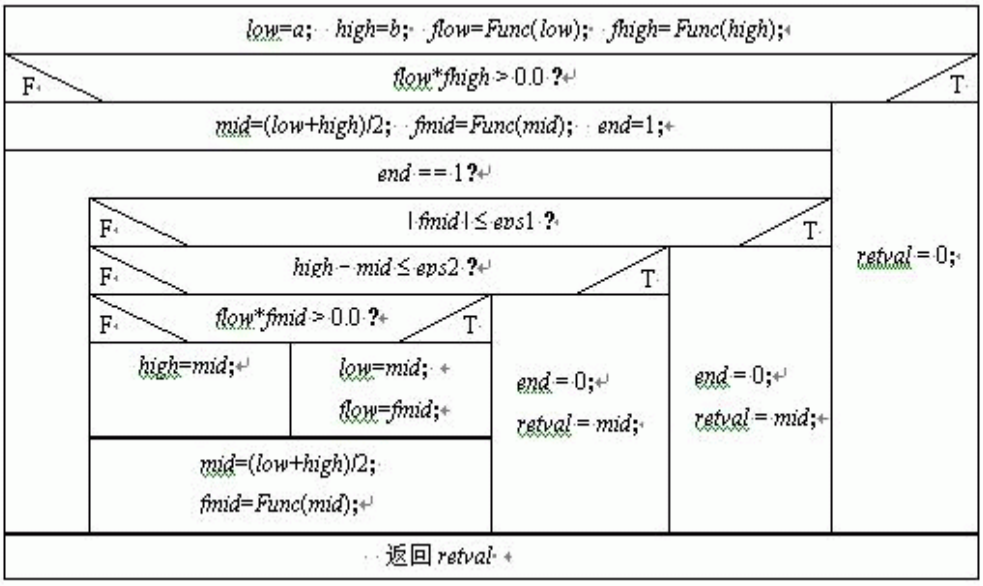
B. 为了实现递归子程序的正确调用，一般使用堆栈来保存每次调用后返回到上一层程序的返回地址、本次递归调用时的形式参数、局部变量等。

C. UNIX 操作系统是 Bell 实验室研制的，用 C 语言写出来的。

【4-7】(1) 结构化的程序流程图：



(2) N-S 图：



(3) 环路复杂性度量 V(G) = 6

【4-8】A. ⑤, B. ②, C. ③, D. ①, E. ③

1960 年 Dijkstra 提倡的结构化程序设计方法，反对（后来改为尽量避免）在程序中使用 GOTO 语句，认为使用顺序、选择、重复等三种基本控制结构进行组合、嵌套，就可以构造出整个程序。这种程序设计方法可以提高程序的生产率，还有利于将来程序的维护。

【4-9】开发规模相同，但复杂性不同的软件，花费的成本和时间会有很大的差异。因此到目前为止，还没有一个软件复杂性度量的方法能够全面、系统地度量任一软件的复杂性，某一种度量方法只偏重于某一方面。所以，用某一种软件复杂性来度量不同类型的程序，所得到的度量值不一定真正反映它们的复杂性。但对同一类型的程序，按某种视点来度量它们的复杂性，其结果还是比较有价值的。

【4-10】K.Magel 从六个方面描述软件复杂性：

① 理解程序的难度；

② 改错及维护程序的难度；

③ 向他人解释程序的难度；

④ 按指定方法修改程序的难度；

⑤ 根据设计文档编写程序的工作量；

⑥ 执行程序时需要资源的程度。

软件复杂性度量模型应遵循的基本原则：

⑴ 软件复杂性与程序大小的关系不是线性的；

⑵ 控制结构复杂的程序较复杂；

⑶ 数据结构复杂的程序较复杂；

⑷ 转向语句使用不当的程序较复杂；

⑸ 循环结构比选择结构复杂，选择结构又比顺序结构复杂；

⑹ 语句、数据、子程序和模块在程序中的次序对软件复杂性都有影响；

⑺ 全程变量、非局部变量较多时程序较复杂；

⑻ 参数按地址传递比按值传递更复杂；

⑼ 函数副作用比显式参数传递更难以琢磨；

⑽ 具有不同作用的变量共用一个名字时较难理解；

⑾ 模块间或过程间联系密切的程序较复杂；

⑿ 嵌套深度越深程序越复杂。

最典型的两种程序复杂性度量的方法中，McCabe 环路复杂性度量就是针对基本原则

(2)制定的度量模型；Halstead 软件科学则是针对程序中操作符和操作数的出现频度而制定的度量模型。

第五章 软件测试

一、复习要求

1. 了解软件测试的目的和原则。

2. 了解软件错误的分类。

3. 了解软件测试的过程和策略。

4. 了解软件测试用例设计的方法，掌握逻辑覆盖、基本路径测试、因果图等测试用例设计方法。

5. 了解程序静态测试的方法。

6. 了解程序调试的概念。

7. 掌握软件测试中的可靠性分析方法

二、内容提要

1. 软件测试基础

(1) 什么是软件测试

软件测试是为了发现错误而执行程序的过程。或者说，软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例（即输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现程序错误的过程。

软件测试在软件生存期中横跨两个阶段：通常在编写出每一个模块之后就对它做必要的测试（称为单元测试）。模块的编写者与测试者是同一个人。编码与单元测试属于软件生存期中的同一个阶段。在这个阶段结束之后，对软件系统还要进行各种综合测试，这是软件生存期的另一个独立的阶段，即测试阶段，通常由专门的测试人员承担这项工作。

(2) 软件测试的目的和原则

Grenford J.Myers 就软件测试目的提出以下观点：

§ 测试是程序的执行过程，目的在于发现错误；

§ 一个好的测试用例在于能发现至今未发现的错误；

§ 一个成功的测试是发现了至今未发现的错误的测试。

设计测试的目标是想以最少的时间和人力系统地找出软件中潜在的各种错误和缺陷。如果我们成功地实施了测试，就能够发现软件中的错误。测试的附带收获是，它能够证明软件的功能和性能与需求说明相符合。此外，实施测试收集到的测试结果数据为可靠性分析提供了依据。

测试不能表明软件中不存在错误，它只能说明软件中存在错误。

软件测试的原则：

① 应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。

不应把软件测试仅仅看作是软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中。坚持在软件开发的各个阶段的技术评审，这样才能在开发过程中尽早发现和预防错误，把出现的错误克服在早期，杜绝某些发生错误的隐患。

② 测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。

测试以前应当根据测试的要求选择测试用例（Test case），用来检验程序员编制的程序，因此不但需要测试的输入数据，而且需要针对这些输入数据的预期输出结果。

③ 程序员应避免检查自己的程序。

程序员应尽可能避免测试自己编写的程序，程序开发小组也应尽可能避免测试本小组开发的程序。如果条件允许，最好建立独立的软件测试小组或测试机构。这点不能与程序的调试（debuging）相混淆。调试由程序员自己来做可能更有效。

④ 在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。

合理的输入条件是指能验证程序正确的输入条件，不合理的输入条件是指异常的，临界的，可能引起问题异变的输入条件。软件系统处理非法命令的能力必须在测试时受到检验。用不合理的输入条件测试程序时，往往比用合理的输入条件进行测试能发现更多的错误。

⑤ 充分注意测试中的群集现象。

在被测程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集性现象，已为许多程序的测试实践所证实。根据这个规律，应当对错误群集的程序段进行重点测试，以提高测试投资的效益。

⑥ 严格执行测试计划，排除测试的随意性。

测试之前应仔细考虑测试的项目，对每一项测试做出周密的计划，包括被测程序的功能、输入和输出、测试内容、进度安排、资源要求、测试用例的选择、测试的控制方式和过程等，还要包括系统的组装方式、跟踪规程、调试规程，回归测试的规定，以及评价标准等。对于测试计划，要明确规定，不要随意解释。

⑦ 应当对每一个测试结果做全面检查。

有些错误的征兆在输出实测结果时已经明显地出现了，但是如果不仔细地全面地检查测试结果，就会使这些错误被遗漏掉。所以必须对预期的输出结果明确定义，对实测的结果仔细分析检查，抓住征侯，暴露错误。

⑧ 妥善保存测试计划，测试用例，出错统计和最终分析报告，为维护提供方便。

(3) 确认和验证的关系

确认（Validation）是一系列的活动和过程，其目的是想证实在一个给定的外部环境中软件的逻辑正确性。它包括需求规格说明的确认和程序的确认，而程序的确认又分为静态确认与动态确认。静态确认一般不在计算机上实际执行程序，而是通过人工分析或者程序正确性证明来确认程序的正确性； 动态确认主要通过动态分析和程序测试来检查程序的执行状态，以确认程序是否有问题。

验证（Verification），则试图证明在软件生存期各个阶段，以及阶段间的逻辑协调性、完备性和正确性。

确认与验证工作都属于软件测试。在对需求理解与表达的正确性、设计与表达的正确性、实现的正确性以及运行的正确性的验证中，任何一个环节上发生了问题都可能在软件测试中表现出来。

(4) 测试信息流

测试信息流如图 5.1 所示。测试过程需要三类输入：

§ 软件配置：包括软件需求规格说明、软件设计规格说明、源代码等；

§ 测试配置：包括测试计划、测试用例、测试驱动程序等；

§ 测试工具：测试工具为测试的实施提供某种服务。例如，测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序、以及驱动测试的工作台等。

测试之后，用实测结果与预期结果进行比较。如果发现出错的数据，就要进行调试。对已经发现的错误进行错误定位和确定出错性质，并改正这些错误，同时修改相关的文档。修正后的文档一般都要经过再次测试，直到通过测试为止。

通过收集和分析测试结果数据，对软件建立可靠性模型。

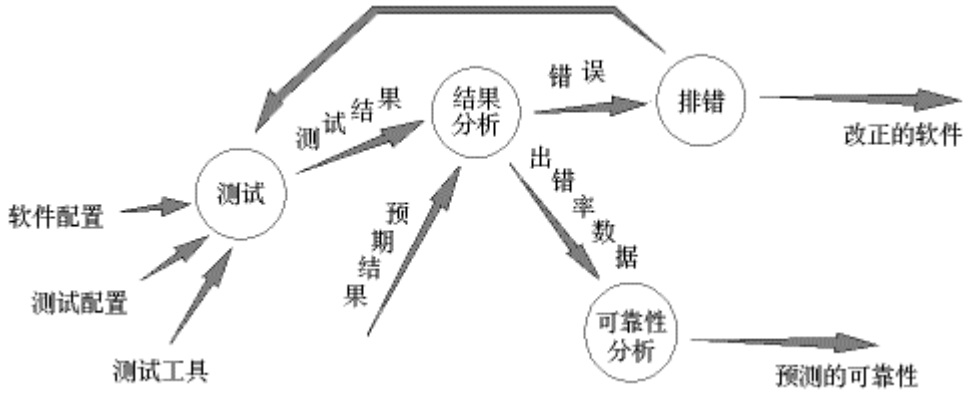


图 5.1 测试信息流

如果测试发现不了错误，那么可以肯定，测试配置考虑得不够细致充分，错误仍然潜伏在软件中。这些错误最终不得不由用户在使用中发现，并在维护时由开发

者去改正。但那时改正错误的费用将比在开发阶段改正错误的费用要高出 40 倍到 60 倍。

(5) 测试与软件开发各阶段的关系

软件开发过程是一个自顶向下，逐步细化的过程，而测试过程则是依相反的顺序安排的 自底向上，逐步集成的过程。低一级测试为上一级测试准备条件。参看图 5.2，首先对每一个程序模块进行单元测试，消除程序模块内部在逻辑上和功能上的错误和缺陷。再对照软件设计进行集成测试，检测和排除子系统（或系统）结构上的错误。随后再对照需求，进行确认测试。最后从系统全体出发，运行系统，看是否满足要求。

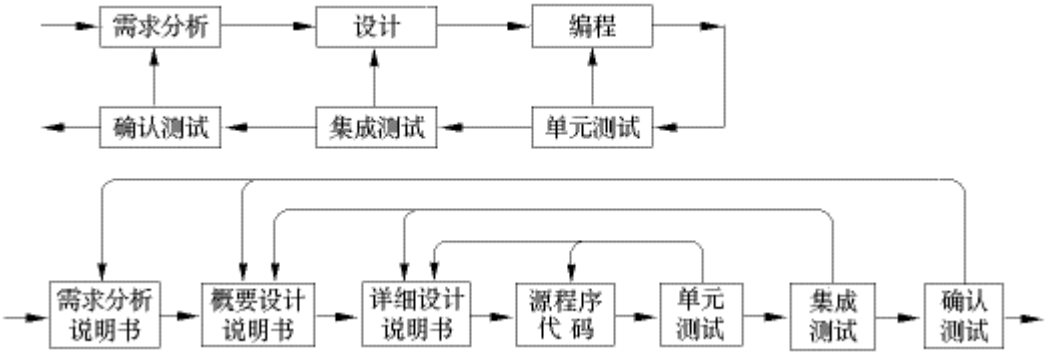


图 5.2 软件测试与软件开发过程的关系

2. 程序错误分类

由于人们对错误有不同的理解和认识，所以目前还没有一个统一的错误分类方法。错误难于分类的原因，一方面是由于一个错误有许多征兆，因而它可以被归入不同的类。另一方面是因为把一个给定的错误归于哪一类，还与错误的来源和程序员的心理状态有关。

(1) 按错误的影响和后果分类

§ 较小错误：只对系统输出有一些非实质性影响。如，输出的数据格式不合要求等。

§ 中等错误：对系统的运行有局部影响。如输出的某些数据有错误或出现冗余。

§ 较严重错误：系统的行为因错误的干扰而出现明显不合情理的现象。比如开出了 0.00 元的支票，系统的输出完全不可信赖。

§ 严重错误：系统运行不可跟踪，一时不能掌握其规律，时好时坏。

§ 非常严重的错误：系统运行中突然停机，其原因不明，无法软启动。

§ 最严重的错误：系统运行导致环境破坏，或是造成事故，引起生命、财产的损失。

(2) 按错误的性质和范围分类

B.Beizer 从软件测试观点出发，把软件错误分为 5 类。

① 功能错误

§ 规格说明错误：规格说明可能不完全，有二义性或自身矛盾。

§ 功能错误：程序实现的功能与用户要求的不一致。这常常是由于规格说明中包含错误的功能、多余的功能或遗漏的功能所致。

§ 测试错误：软件测试的设计与实施发生错误。软件测试自身也可能发生错误。

§ 测试标准引起的错误：对软件测试的标准要选择适当，若测试标准太复杂，则导致测试过程出错的可能就大。

② 系统错误

§ 外部接口错误：外部接口指如终端、打印机、通信线路等系统与外部环境通信的手段。所有外部接口之间，人与机器之间的通信都使用形式的或非形式的专门协议。如果协议有错，或太复杂，难以理解，致使在使用中出错。此外还包括对输入 / 输出格式错误理解，对输入数据不合理的容错等等。

§ 内部接口错误：内部接口指程序之间的联系。它所发生的错误与程序内实现的细节有关。例如，设计协议错、输入 / 输出格式错、数据保护不可靠、子程序访问错等。

§ 硬件结构错误：这类错误在于不能正确地理解硬件如何工作。例如，忽视或错误地理解分页机构、地址生成、通道容量、I / O 指令、中断处理、设备初始化和启动等而导致的出错。

§ 操作系统错误：这类错误主要是由于不了解操作系统的工作机制而导致出错。。当然，操作系统本身也有错误，但是一般用户很难发现这种错误。

§ 软件结构错误：由于软件结构不合理或不清晰而引起的错误。这种错误通常与系统的负载有关，而且往往在系统满载时才出现。这是最难发现的一类错误。例如，错误地设置局部参数或全局参数；错误地假定寄存器与存储器单元初始化了；错误地假定不会发生中断而导致不能封锁或开中断；错误地假定程序可以绕过数据的内部锁而导致不能关闭或打开内部锁；错误地假定被调用子程序常驻内存或非常驻内存等等，都将导致软件出错。

§ 控制与顺序错误：这类错误包括：忽视了时间因素而破坏了事件的顺序；猜测事件出现在指定的序列中；等待一个不可能发生的条件；漏掉先决条件；规定

错误的优先级或程序状态；漏掉处理步骤；存在不正确的处理步骤或多余的处理步骤等。

§ 资源管理错误：这类错误是由于不正确地使用资源而产生的。例如，使用未经获准的资源；使用后未释放资源；资源死锁；把资源链接在错误的队列中等等。

③ 加工错误

§ 算术与操作错误：指在算术运算、函数求值和一般操作过程中发生的错误。包括：数据类型转换错；除法溢出；错误地使用关系比较符；用整数与浮点数做比较等。

§ 初始化错误：典型的错误有：忘记初始化工作区，忘记初始化寄存器和数据区；错误地对循环控制变量赋初值；用不正确的格式，数据或类型进行初始化等等。

§ 控制和次序错误：这类错误与系统级同名错误类似，但它是局部错误。包括：遗漏路径；不可达到的代码；不符合语法的循环嵌套；循环返回和终止的条件不正确；漏掉处理步骤或处理步骤有错等。

§ 静态逻辑错误：这类错误主要包括：不正确地使用 CASE 语句；在表达式中使用不正确的否定（例如用“＞”代替“＜”的否定）；对情况不适当地分解与组合；混淆“或”与“异或”等。

④ 数据错误

§ 动态数据错误：动态数据是在程序执行过程中暂时存在的数据。各种不同类型的动态数据在程序执行期间将共享一个共同的存储区域，若程序启动时对这个区域未初始化，就会导致数据出错。由于动态数据被破坏的位置可能与出错的位置在距离上相差很远，因此要发现这类错误比较困难。

§ 静态数据错误：静态数据在内容和格式上都是固定的。它们直接或间接地出现在程序或数据库中。由编译程序或其它专门程序对它们做预处理。这是在程序执行前防止静态错误的好办法，但预处理也会出错。

§ 数据内容错误：数据内容是指存储于存储单元或数据结构中的位串、字符串或数字。数据内容本身没有特定的含义，除非通过硬件或软件给予解释。数据内容错误就是由于内容被破坏或被错误地解释而造成的错误。

§ 数据结构错误：数据结构是指数据元素的大小和组织形式。在同一存储区域中可以定义不同的数据结构。数据结构错误主要包括结构说明错误及把一个数据结构误当做另一类数据结构使用的错误。这是更危险的错误。

§ 数据属性错误：数据属性是指数据内容的含义或语义。例如，整数、字符串、子程序等等。数据属性错误主要包括：对数据属性不正确地解释，比如错把整数当实数，允许不同类型数据混合运算而导致的错误等。

⑤ 代码错误

主要包括：语法错误；打字错误；对语句或指令不正确理解所产生的错误。

(3) 按软件生存期阶段分类

Good enough－Gerhart 分类方法把软件的逻辑错误按生存期不同阶段分为 4 类。

① 问题定义（需求分析）错误

它们是在软件定义阶段，分析员研究用户的要求后所编写的文档中出现的错误。换句话说，这类错误是由于问题定义不满足用户的要求而导致的错误。

② 规格说明错误

这类错误是指规格说明与问题定义不一致所产生的错误。它们又可以细分成：

§ 不一致性错误：规格说明中功能说明与问题定义发生矛盾。

§ 冗余性错误：规格说明中某些功能说明与问题定义相比是多余的。

§ 不完整性错误：规格说明中缺少某些必要的功能说明。

§ 不可行错误：规格说明中有些功能要求是不可行的。

§ 不可测试错误：有些功能的测试要求是不现实的。

③ 设计错误

这是在设计阶段产生的错误，它使系统的设计与需求规格说明中的功能说明不相符。它们又可以细分为：

§ 设计不完全错误：某些功能没有被设计，或设计得不完全。

§ 算法错误：算法选择不合适。主要表现为算法的基本功能不满足功能要求、算法不可行或者算法的效率不符合要求。

§ 模块接口错误：模块结构不合理；模块与外部数据库的界面不一致，模块之间的界面不一致。

§ 控制逻辑错误：控制流程与规格说明不一致；控制结构不合理。

§ 数据结构错误：数据设计不合理；与算法不匹配；数据结构不满足规格说明要求。

④ 编码错误

编码过程中的错误是多种多样的，大体可归为以下几种：数据说明错、数据使用错、计算错、比较错、控制流错、界面错、输入 / 输出错，及其它的错误。

在不同的开发阶段，错误的类型和表现形式是不同的，故应当采用不同的方法和策略来进行检测。

3. 软件测试的过程与策略

测试过程按 4 个步骤进行，即单元测试、组装测试、确认测试和系统测试。图 5.3 显示出软件测试经历的 4 个步骤。单元测试集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。然后，进行集成测试，根据设计规定的软件体系结构，把已测试过的模块组装起来，在组装过程中，检查程序结构组装的正确性。确认测试则是要检查已实现的软件是否满足了需求规格说明中确定了的各种需求，以及软件配置是否完全、正确。最后是系统测试，把已经经过确认的软件纳入实际运行环境中，与其它系统成份组合在一起进行测试。严格地说，系统测试已超出了软件工程的范围。

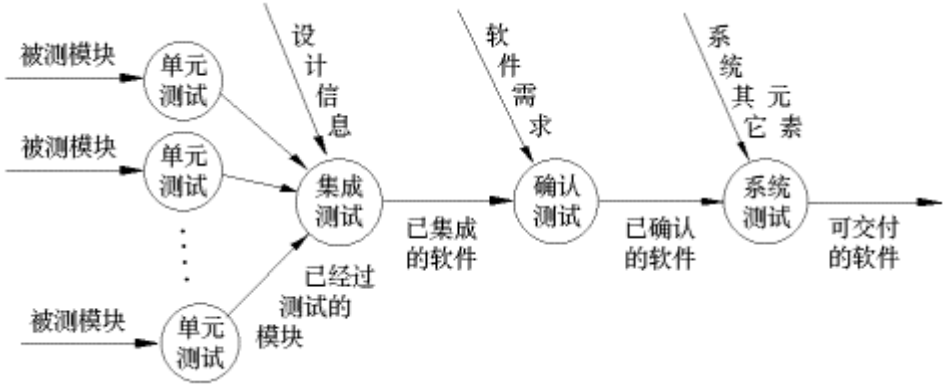


图 5.3 软件测试的过程

(1) 单元测试

单元测试针对程序模块，进行正确性检验的测试。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

① 单元测试的内容

§ 模块接口测试 ：对通过被测模块的数据流进行测试。为此，对模块接口，包括参数表、调用子模块的参数、全程数据、文件输入 / 输出操作都必须检查。

§ 局部数据结构测试 ：设计测试用例检查数据类型说明、初始化、缺省值等方面的问题，还要查清全程数据对模块的影响。

§ 路径测试 ： 选择适当的测试用例，对模块中重要的执行路径进行测试。对基本执行路径和循环进行测试可以发现大量的路径错误。

§ 错误处理测试 ：检查模块的错误处理功能是否包含有错误或缺陷。例如，是否拒绝不合理的输入；出错的描述是否难以理解、是否对错误定位有误、是否出错原因报告有误、是否对错误条件的处理不正确；在对错误处理之前错误条件是否已经引起系统的干预等。

§ 边界测试 ：要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。

此外，如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。这类信息对进行性能评价是十分有用的。

② 单元测试的步骤

通常单元测试在编码阶段进行。在源程序代码编制完成，经过评审和验证，确认没有语法错误之后，就开始进行单元测试的测试用例设计。利用设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其它模块。这些辅助模块分为两种：

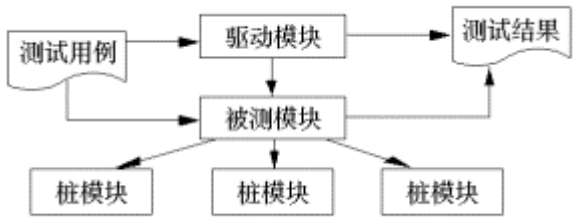


图 5.4 单元测试的测试环境

§ 驱动模块：相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后输出实测结果。

§ 桩模块：用以代替被测模块调用的子模块。桩模块可以做少量的数据操作，不需要把子模块所有功能都带进来，但不允许什么事情也不做。

被测模块、与它相关的驱动模块及桩模块共同构成了一个“测试环境”，见图 5.4。

如果一个模块要完成多种功能，且以程序包或对象类的形式出现，例如 Ada 中的包，MODULA 中的模块，C++中的类。这时可以将这个模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作，对关键模块还要做性能测试。对支持某些标准规程的程序，更要着手进行互联测试。有人把这种情况特别称为模块测试，以区别单元测试。

(2) 集成测试

在单元测试的基础上，需要将所有模块按照设计要求组装成为系统。这时需要考虑：

§ 在把各个模块连接起来的时侯，穿越模块接口的数据是否会丢失；

§ 一个模块的功能是否会对另一个模块的功能产生不利的影响；

§ 各个子功能组合起来，能否达到预期要求的父功能；

§ 全局数据结构是否有问题；

§ 单个模块的误差累积起来，是否会放大，从而达到不能接受的程度。

§ 单个模块的错误是否会导致数据库错误。

选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号的次序和测试的次序、以及生成测试用例的费用和调试的费用。通常，把模块组装成为系统的方式有两种方式：

① 一次性集成方式

它是一种非增殖式集成方式。也叫做整体拼装。使用这种方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。

由于程序中不可避免地存在涉及模块间接口、全局数据结构等方面的问题，所以一次试运行成功的可能性并不很大。

② 增殖式集成方式

又称渐增式集成方式。首先对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统，在组装的过程中边连接边测试，以发现连接过程中产生的问题。最后通过增殖逐步组装成为要求的软件系统。

§ 自顶向下的增殖方式：将模块按系统程序结构，沿控制层次自顶向下进行集成。由于这种增殖方式在测试过程中较早地验证了主要的控制和判断点。在一个功能划分合理的程序结构中，判断常出现在较高的层次，较早就能遇到。如果主要控制有问题，尽早发现它能够减少以后的返工。

§ 自底向上的增殖方式：从程序结构的最底层模块开始组装和测试。因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块（包括子模块的所有下属模块）已经组装并测试完成，所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以直接运行子模块得到。

③ 混合增殖式测试：自顶向下增殖的方式和自底向上增殖的方式各有优缺点。自顶向下增殖方式的缺点是需要建立桩模块。要使桩模块能够模拟实际子模块的功能将是十分困难的。同时涉及复杂算法和真正输入／输出的模块一般在底层，它们是最容易出问题的模块，到组装和测试的后期才遇到这些模块，一旦发现问题，导致过多的回归测试。而自顶向下增殖方式的优点是能够较早地发现在主要控制方面的问题。自底向上增殖方式的缺点是“程序一直未能做为一个实体存在，直到最后一个模块加上去后才形成一个实体”。就是说，在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。但这种方式的优点是不需要桩模块，而建立驱动模块一般比建立桩模块容易，同时由于涉及到复杂算法和真正输入／输出的模块最先得到组装和测试，可以把最容易出问题的部分在早期解决。此外自底向上增殖的方式可以实施多个模块的并行测试。 有鉴于此，通常是把以上两种方式结合起来进行组装和测试。

§ 衍变的自顶向下的增殖测试：它的基本思想是强化对输入／输出模块和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增殖测试。

§ 自底向上-自顶向下的增殖测试：它首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

§ 回归测试：这种方式采取自顶向下的方式测试被修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否适配。

(3) 确认测试

确认测试又称有效性测试。它的任务是验证软件的有效性，即验证软件的功能和性能及其它特性是否与用户的要求一致。在软件需求规格说明书描述了全部用户可见的软件属性，其中有一节叫做有效性准则，它包含的信息就是软件确认测试的基础。

在确认测试阶段需要做的工作如图 5.5 所示。首先要进行有效性测试以及软件配置复审，然后进行验收测试和安装测试，在通过了专家鉴定之后，才能成为可交付的软件。

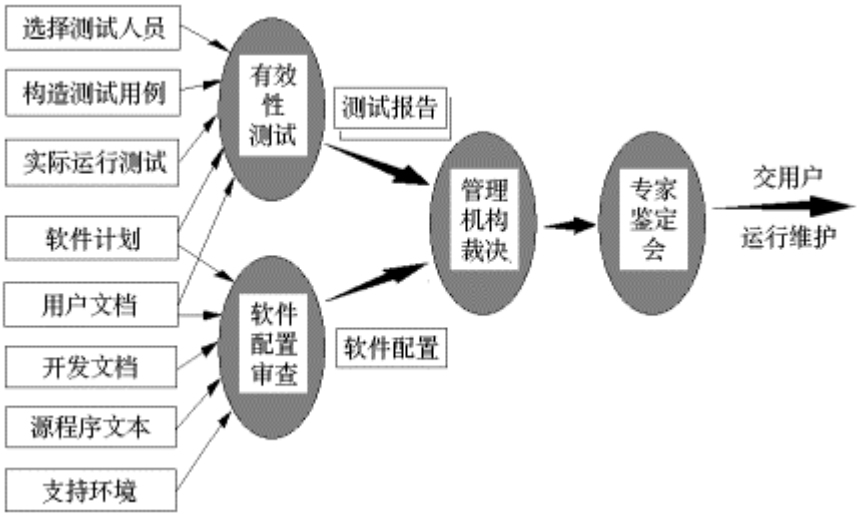


图 5.5 确认测试的步骤

① 进行有效性测试（功能测试） 有效性测试是在模拟的环境（可能就是开发的环境）下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。为此，需要首先制定测试计划，规定要做测试的种类。还需要制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求都能达到，所有的文档都是正确且便于使用。同时，对其它软件需求，例如可移植性、兼容性、出错自动恢复、可维护性等，也都要进行测试，确认是否满足。

② 软件配置复查

软件配置复查的目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，具有维护阶段所必需的细节，而且已经编排好分类的目录。

除了按合同规定的内容和要求，由人工审查软件配置之外，在确认测试的过程中，应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误，并且适当地补充和改正。

③ 验收测试

在通过了系统的有效性测试及软件配置审查之后，就应开始系统的验收测试。验收测试是以用户为主的测试。软件开发人员和 QA（质量保证）人员也应参加。由用户参加设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果。一般使用生产中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

④ α 测试和 β 测试

在软件交付使用之后，用户将如何实际使用程序，对于开发者来说是无法预测的。因为用户在使用过程中常常会发生对使用方法的误解、异常的数据组合、以及产生对某些用户来说似乎是清晰的但对另一些用户来说却难以理解的输出等等。

如果软件是为多个用户开发的产品的时候，让每个用户逐个执行正式的验收测试是不切实际的。很多软件产品生产者采用一种称之为 α 测试和 β 测试的测试方法，以发现可能只有最终用户才能发现的错误。

α 测试是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。这是在受控制的环境下进行的测试。α 测试的目的是评价软件产品的 FURPS（即功能、可使用性、可靠性、性能和支持）。尤其注重产品的界面和特色。α 测试人员是除开产品开发人员之外首先见到产品的人，他们提出的功能和修改意见是特别有价值的。α 测试可以从软件产品编码结束之时开始，或在模块（子系统）测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。有关的手册（草稿）等应事先准备好。 β 测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。与 α 测试不同的是，开发者通常不在测试现场。因而，β 测试是在开发者无法控制的环境下进行的软件现场应用。在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告，开发者在综合用户的报告之后，做出修改，最后将软件产品交付给全体用户使用。β 测试主要衡量产品的 FURPS。着重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本也应该在此阶段完全定稿。由于 β 测试的主要目标是测试可支持性，所以 β 测试应尽可能由主持产品发行的人员来管理。

(4) 系统测试

所谓系统测试，是将通过确认测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其它系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试。

系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统定义不符合或与之矛盾的地方。系统测试的测试用例应根据需求分析规格说明来设计，并在实际使用环境下来运行。

4. 测试用例设计

(1) 测试方法概述

软件测试的种类大致可以分为人工测试和基于计算机的测试。而基于计算机的测试由可以分为白盒测试和黑盒测试。

① 黑盒测试

根据软件产品的功能设计规格，在计算机上进行测试，以证实每个实现了的功能是否符合要求。这种测试方法就是黑盒测试。黑盒测试意味着测试要在软件的接口处进行。就是说，这种方法是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求分析规格说明，检查程序的功能是否符合它的功能说明。

用黑盒测试发现程序中的错误，必须在所有可能的输入条件和输出条件中确定测试数据，来检查程序是否都能产生正确的输出。

② 白盒测试

根据软件产品的内部工作过程，在计算机上进行测试，以证实每种内部操作是否符合设计规格要求，所有内部成分是否已经过检查。这种测试方法就是白盒测试。白盒测试把测试对象看做一个打开的盒子，允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。 不论是黑盒测试，还是白盒测试，都不可能把所有可能的输入数据都拿来进行所谓的穷举测试。因为可能的测试输入数据数目往往达到天文数字。下面让我们看两个例子。

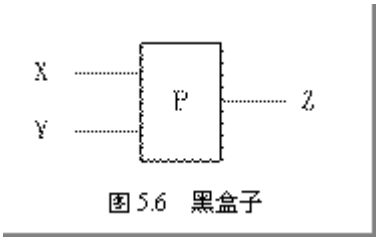


图 5.6 黑盒子

假设一个程序 P 有输入 X 和 Y 及输出 Z，参看图 5.6。在字长为 32 位的计算机上运行。如果 X、Y 只取整数，考虑把所有的 X、Y 值都做为测试数据，按黑盒测试方法进行穷举测试，力图全面、无遗漏地“挖掘”出程序中的所有错误。这样做可能采用的测试数据组（Xi, Yi）的最大可能数目为：232×232＝264。如果程序 P 测试一组 X、Y 数据需要 1 毫秒，且一天工作 24 小时，一年工作 365 天，要完成 264 组测试，需要 5 亿年。

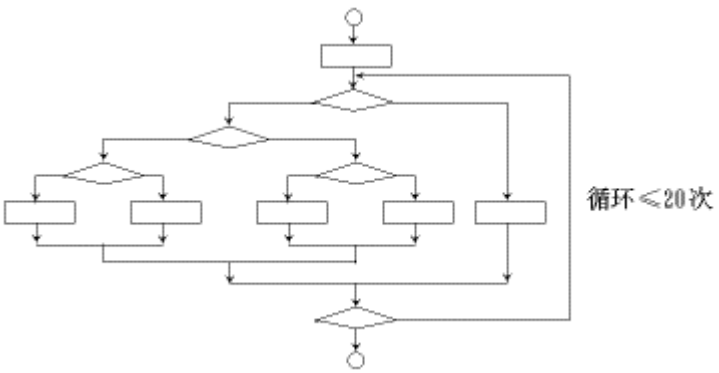


图 5.7 白盒测试中的穷举测试

而对一个具有多重选择和循环嵌套的程序，不同的路径数目也可能是天文数字。设给出一个如图 5.7 所示的小程序的流程图，其中包括了一个执行达 20 次的循环。那么它所包含的不同执行路径数高达 520（＝1013）条，若要对它进行穷举测试，覆盖所有的路径。假使测试程序对每一条路径进行测试需要 1 毫秒，同样假定一天工作 24 小时，一年工作 365 天， 那么要想把如图 5.7 所示的小程序的所有路径测试完，则需要 3170 年。

以上的分析表明，实行穷举测试，由于工作量过大，实施起来是不现实的。任何软件开发项目都要受到期限、费用、人力和机时等条件的限制，尽管为了充分揭露程序中所有隐藏错误，需要针对所有可能的数据进行测试，但事实告诉我们，这样做是不可能的。 软件工程的总目标是充分利用有限的人力、物力资源，高效率、高质量、低成本地完成软件开发项目。在测试阶段既然穷举测试不可行，为了节省时间和资源，提高测试效率，就必须要从数量极大的可用测试用例中精心地挑选少量的测试数据，使得采用这些测试数据能够达到最佳的测试效果，能够高效率地把隐藏的错误揭露出来。

(2) 逻辑覆盖

逻辑覆盖是以程序内部的逻辑结构为基础的设计测试用例的技术。属白盒测试。这一方法要求测试人员对程序的逻辑结构有清楚的了解，甚至要能掌握源程序的所有细节。由于覆盖测试的目标不同，逻辑覆盖又可分为：语句覆盖、判定覆盖、判定－条件覆盖、条件组合覆盖及路径覆盖。

① 语句覆盖 ：语句覆盖就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。这种覆盖又称为点覆盖，它使得程序中每个可执行语句都得到执行，但它是最弱的逻辑覆盖准，效果有限，必须与其它方法交互使用。

② 判定覆盖 ：判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。判定覆盖又称为分支覆盖。

判定覆盖只比语句覆盖稍强一些，但实际效果表明，只是判定覆盖，还不能保证一定能查出在判断的条件中存在的错误。因此，还需要更强的逻辑覆盖准则去检验判断内部条件。

③ 条件覆盖 ：条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。

条件覆盖深入到判定中的每个条件，但可能不能满足判定覆盖的要求。

④ 判定－条件覆盖 ：判定－条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断本身的所有可能判断结果至少执行一次。换言之，即是要求各个判断的所有可能的条件取值组合至少执行一次。

判定－条件覆盖有缺陷。从表面上来看，它测试了所有条件的取值。但是事实并非如此。往往某些条件掩盖了另一些条件。会遗漏某些条件取值错误的情况。为彻底地检查所有条件的取值，需要将判定语句中给出的复合条件表达式进行分解，形成由多个基本判定嵌套的流程图。这样就可以有效地检查所有的条件是否正确了。

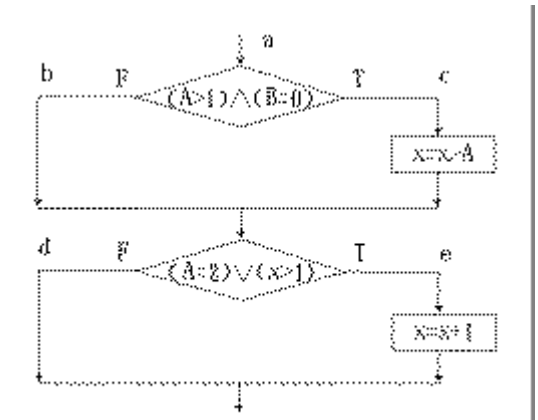


图 5.8(a) 复合判定的例子

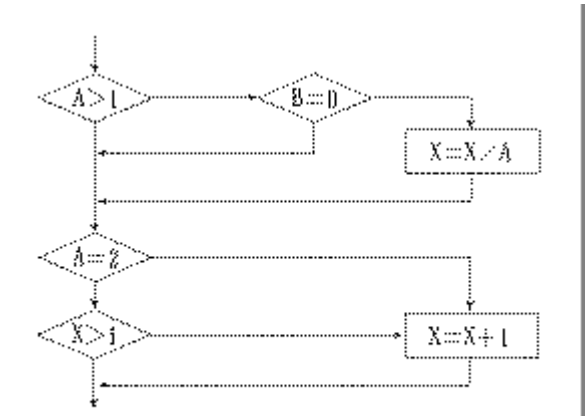


图 5.8(b) 改为单个条件判定的嵌套结构的例子

⑤ 多重条件覆盖 ：多重条件覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。

这是一种相当强的覆盖准则，可以有效地检查各种可能的条件取值的组合是否正确。它不但可覆盖所有条件的可能取值的组合，还可覆盖所有判断的可取分支，但可能有的路径会遗漏掉。测试还不完全。

⑥路径测试 ：路径测试就是设计足够的测试用例，覆盖程序中所有可能的路径。这是最强的覆盖准则。但在路径数目很大时，真正做到完全覆盖是很困难的，必须把覆盖路径数目压缩到一定限度。下面我们做一分析。

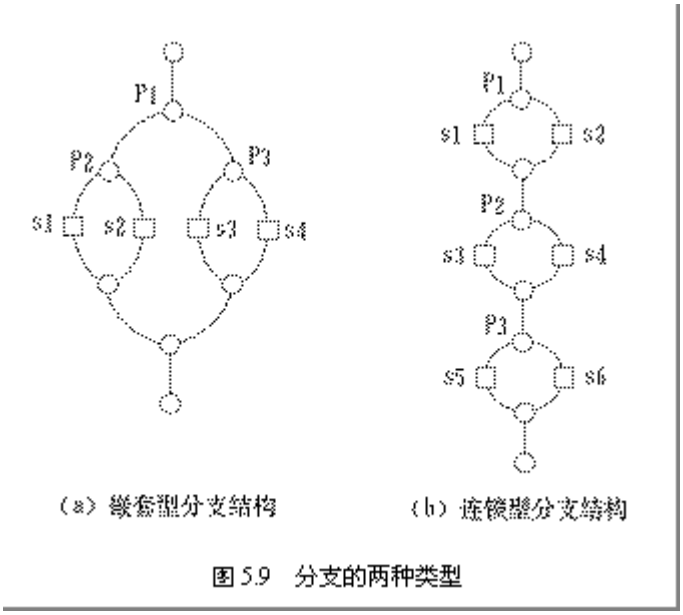


图 5.9 分支的两种类型

(3) 关于控制结构测试的一些讨论

① 分支结构的路径数

当程序中判定多于一个时，形成的分支结构可以分为两类：嵌套型分支结构和连锁型分支结构。如图 5.9 所示。对于嵌套型分支结构，若有 n 个判定语句，则需要 n+1 个测试用例；但对连锁型分支结构，若有 n 个判定语句，则需要有 2n 个测试用例，去覆盖它的 2n 条路径。当 n 较大时将无法测试。

为减少测试用例的数目，可采用试验设计法，抽取部分路径进行测试。由于抽样服从均匀分布，因此，在假定各条路径的重要性相同，或暂不明确各条路径的重要性的情况下可以做到均匀抽样。如果明确了各条路径的重要性，还可以采取加权的办法，筛选掉部分路径，再用如下的措施进行抽样。具体步骤如下： ⅰ）设耦合型分支结构中有 n 个判定，计算满足关系式 n+1≤2m 的最小自然数 m；

ⅱ）设 t = 2m，取正交表 Lt，并利用它设计测试数据。

例如，一个耦合型分支结构中有三个判定语句 P1，P2，P3。它全部路径是 23＝8 条。先计算 3+1≤2m = t 的 t，得 t = 4。取正交表 L4，如图 5.10 (a) 所示，把每一列当做一个判定，每一行当做可取的测试用例，则正交表 L4 最多可取三个判定，分别代之以 P1，P2，P3。判定 P1，P2，P3 的取假分支和取真分支分别记作 S1、S2；S3、S4；S5、S6，用各个判定的取假分支取代正交表 L4 中的“0”，用取真分支取代正交表中的“1”，就建立起一个测试路径矩阵，如图 5.10 (b) 所示。这样，测试路径数目从 23＝8 条减少到 3＋1＝4 条。

L4	1	2	3
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	0

用例	P1	P2	P3	路径
1	S1	S3	S5	S1-S3-S5
2	S2	S3	S6	S2-S3-S6
3	S1	S4	S6	S1-S4-S6
4	S2	S4	S5	S2-S4-S6

图 5.10（a）正交表 图 5.10（b）路径抽样矩阵

② 条件测试的策略

程序中的条件分为简单条件和复合条件。简单条件是一个布尔变量或一个关系表达式（可加前缀 NOT），复合条件由简单条件通过逻辑运算符（AND、OR、NOT）和括号连接而成。如果条件出错，至少是条件中某一成分有错。条件中可能的出错类型有：布尔运算符错、布尔变量错、布尔括号错、关系运算符错、算术表达式错。

如果在一个判定的复合条件表达式中每个布尔变量和关系运算符最多只出现一次，而且没有公共变量，应用一种称之为 BRO（分支与关系运算符）的测试法可以发现多个布尔运算符或关系运算符错，以及其它错误。

BRO 策略引入条件约束的概念。设有 n 个简单条件的复合条件 C，其条件约束为 D =（D1, D2, …, Dn），其中 Di（1≤i≤n）是条件 C 中第 i 个简单条件的输出约束。如果在 C 的执行过程中，其每个简单条件的输出都满足 D 中对应的约束，则称条件 C 的条件约束 D 由 C 的执行所覆盖。特别地，布尔变量或布尔表达式的输出约束必须是真（t）或假（f）；关系表达式的输出约束为符号>、=、<。

§ 设条件为 C1 : B1 & B2

其中 B1、B2 是布尔变量，C1 的输出约束为（D1, D2），在此，D1 和 D2 或为 t 或为 f。则（t, f）是 C1 可能的一个约束。覆盖此约束的测试（一次运行）将令 B1 为 t，B2 为 f。BRO 策略要求对 C1 的可能约束集合 {（t, t）,（f, t）,（t, f）} 中的每一个，分别设计一组测试用例。如果布尔运算符有错，这三组测试用例的运行结果必有一组导致 C1 失败。

§ 设条件为 C2 : B1 &（E3 = E4）

其中 B1 是布尔表达式，E3 和 E4 是算术表达式，C2 的输出约束为（D1, D2），在此，D1 或为 t 或为 f；D2 则是 <、= 或 >。因此，只有 D2 与 C1 中 D2 的不同，可以修改 C1 的约束集合 {（t, t）,（f, t）,（t, f）}，导出 C2 的约束集合。因为在（E3 = E4）中，"t" 相当于 "="，"f" 相当于 "<" 或 ">"，则 C2 的约束集合为 {（t, =）,（f, =）,（t, <）,（t, >）}。据此设计 4 组测试用例，检查 C2 中可能的布尔或关系运算符中的错误。

§ 设条件为 C3 :（E1 > E2）&（E3 = E4）

其中 E1、E2、E3、E4 都是算术表达式，C3 的输出约束为（D1, D2），在此，D1 和 D2 的约束均为 <、=、>。C3 中只有 D1 与 C2 中的 D1 不同，可以修改 C2 的约束集合 {（t, =），（f, =），（t, <），（t, >）}，导出 C3 的约束集合。因为在（E1 > E2）中，"t" 相当于 ">"，"f" 相当于 "<" 或 "="，则 C3 的约束集合为 {（>, =），（<, =），（=, =），（>, <），（>, >）}。根据这个约束集合设计测试用例，就能够检测 C3 中的关系运算符中的错误。

③ 循环测试

循环分为 4 种不同类型：简单循环、连锁循环、嵌套循环和非结构循环，见图 5.11。

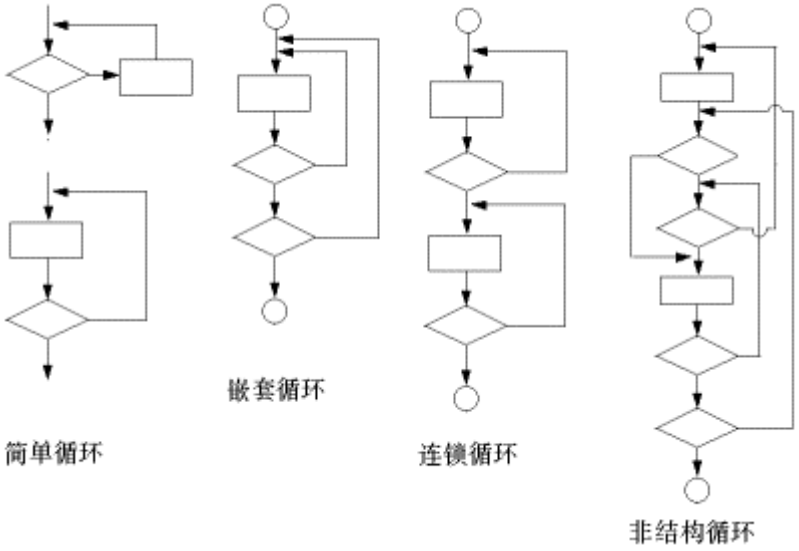


图 5.11 循环的分类

对于简单循环，测试应包括以下几种。其中的 n 表示循环允许的最大次数。

§ 零次循环：从循环入口直接跳到循环出口。

§ 一次循环：查找循环初始值方面的错误。

§ 二次循环：检查在多次循环时才能暴露的错误。

§ m 次循环：此时的 m<n，也是检查在多次循环时才能暴露的错误。

§ 最大次数循环、比最大次数多一次的循环、比最大次数少一次的循环。

对于嵌套循环，不能将简单循环的测试方法简单地扩大到嵌套循环，因为可能的测试数目将随嵌套层次的增加呈几何倍数增长。这可能导致一个天文数字的测试数目。下面给出一种有助于减少测试数目的测试方法。

§ 除最内层循环外，从最内层循环开始，置所有其它层的循环为最小值；

§ 对最内层循环做简单循环的全部测试。测试时保持所有外层循环的循环变量为最小值。另外，对越界值和非法值做类似的测试。

§ 逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环的循环变量取最小值，所有其它嵌套内层循环的循环变量取“典型”值。

§ 反复进行，直到所有各层循环测试完毕。

§ 对全部各层循环同时取最小循环次数，或者同时取最大循环次数。对于后一种测试，由于测试量太大，需人为指定最大循环次数。 对于连锁循环，要区别两种情况。如果各个循环互相独立，则连锁循环可以用与简单循环相同的方法进行测试。例如，有两个循环处于连锁状态，则前一个循环的循环变量的值就可以做为后一个循环的初值。但如果几个循环不是互相独立的，则需要使用测试嵌套循环的办法来处理。

对于非结构循环，应该使用结构化程序设计方法重新设计测试用例。

(4) 基本路径测试

如果把覆盖的路径数压缩到一定限度内，例如，程序中的循环体只执行零次和一次，就成为基本路径测试。它是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。 设计出的测试用例要保证在测试中，程序的每一个可执行语句至少要执行一次。

① 程序的控制流图

控制流图是描述程序控制流的一种图示方法。基本控制构造的图形符号如图 5.12 所示。符号○称为控制流图的一个结点，一组顺序处理框可以映射为一个单一的结点。控制流图中的箭头称为边，它表示了控制流的方向，在选择或多分支结构中分支的汇聚处，即使没有执行语句也应该有一个汇聚结点。边和结点圈定的区域叫做区域，当对区域计数时，图形外的区域也应记为一个区域。



图 5.12 控制流图的各种图形符号

如果判定中的条件表达式是复合条件时，即条件表达式是由一个或多个逻辑运算符（OR，AND，NAND，NOR）连接的逻辑表达式，则需要改复合条件的判定为一系列只有单个条件的嵌套的判定。例如对应图 5.13 (a) 的复合条件的判定，应该画成如图 5.13 (b) 所示的控制流图。 条件语句 if a OR b 中条件 a 和条件 b 各有一个只有单个条件的判定结点。

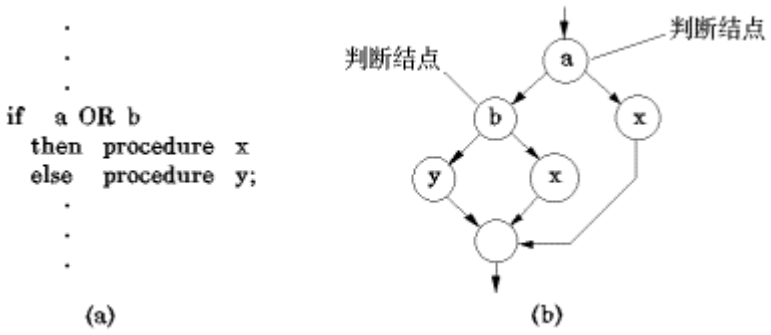


图 5.13 复合逻辑下的控制流图

② 计算程序环路复杂性

进行程序的基本路径测试时，程序的环路复杂性给出了程序基本路径集合中的独立路径条数，这是确保程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。 所谓独立路径，是指包括一组以前没有处理的语句或条件的一条路径。如在图 5.14(b)所示的控制流图中，一组独立的路径是

path1：1 - 11

path2：1 - 2 - 3 - 4 - 5 - 10 - 1 - 11

path3：1 - 2 - 3 - 6 - 8 - 9 - 10 - 1 - 11

path4：1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11

路径 path1，path2，path3，path4 组成了图 5.14 (b) 所示控制流图的一个基本路径集。只要设计出的测试用例能够确保这些基本路径的执行，就可以使得程序中的每个可执行语句至少执行一次，每个条件的取真和取假分支也能得到测试。基本路径集不是唯一的，对于给定的控制流图，可以得到不同的基本路径集。

(a) 程序流程图 (b) 控制流图

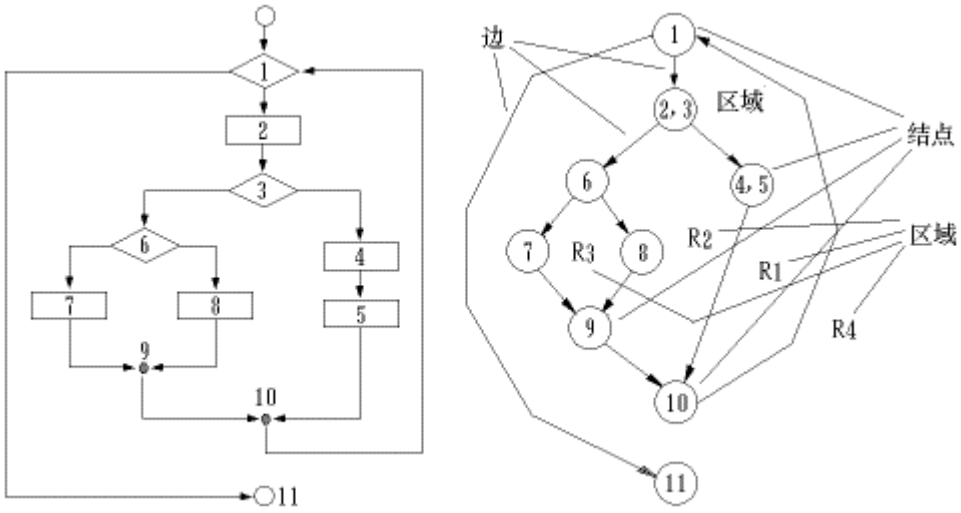


图 5.14 程序流程图与对应的控制流图

通常环路复杂性可用以下三种方法求得。

§ 将环路复杂性定义为控制流图中的区域数。

§ 设 E 为控制流图的边数，N 为图的结点数，则定义环路复杂性为 V(G)＝E－N＋2。

§ 若设 P 为控制流图中的判定结点数，则有 V(G)＝P＋1。

因为图 5.14(b)所示控制流图有 4 个区域。其环路复杂性为 4。 它是构成基本路径集的独立路径数的上界。可以据此得到应该设计的测试用例的数目。

③ 导出测试用例

利用逻辑覆盖方法生成测试用例，确保基本路径集中每条路径的执行。

(5) 等价类划分

等价类划分是一种典型的黑盒测试方法。使用这一方法时，完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例。由于不可能用所有可以输入的数据来测试程序，而只能从全部可供输入的数据中选择一个子集进行测试。如何选择适当的子集，使其尽可能多地发现错误。解决的办法之一就是等价类划分。

首先把数目极多的输入数据（有效的和无效的）划分为若干等价类。所谓等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。并合理地假定：测试某等价类的代表值就等价于对这一类其它值的测试。因此，我们可以把全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据做为测试的输入条件，就可用少量代表性测试数据，取得较好的测试效果。

等价类的划分有两种不同的情况：

§ 有效等价类：是指对于程序规格说明来说，是合理的，有意义的输入数据构成的集合。利用它，可以检验程序是否实现了规格说明预先规定的功能和性能。

§ 无效等价类：是指对于程序规格说明来说，是不合理的，无意义的输入数据构成的集合。利用它，可以检查程序中功能和性能的实现是否有不符合规格说明要求的地方。

在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。软件不能都只接收合理的数据，还要经受意外的考验，接受无效的或不合理的数据，这样获得的软件才能具有较高的可靠性。划分等价类的原则如下：

§ 按区间划分：如果可能的输入数据属于一个取值范围或值的个数限制范围，则可以确立一个有效等价类和两个无效等价类。 § 按数值划分：如果规定了输入数据的一组值，而且程序要对每个输入值分别进行处理。则可为每一个输入值确立一个有效等价类，此外针对这组值确立一个无效等价类，它是所有不允许的输入值的集合。

§ 按数值集合划分：如果可能的输入数据属于一个值的集合，或者须满足“必须如何”的条件，这时可确立一个有效等价类和一个无效等价类。

§ 按限制条件或规则划分：如果规定了输入数据必须遵守的规则或限制条件，则可以确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

② 确立测试用例 在确立了等价类之后，建立等价类表，列出所有划分出的等价类:

再从划分出的等价类中按以下原则选择测试用例：

§ 设计尽可能少的测试用例，覆盖所有的有效等价类；

§ 针对每一个无效等价类，设计一个测试用例来覆盖它。

(6) 边界值分析

人们从长期的测试工作经验得知，大量的错误是发生在输入或输出范围的边界上，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。
比如，在做三角形计算时，要输入三角形的三个边长：A、B 和 C。 我们应注意到这三个数值应当满足 A＞0、B＞0、C＞0、A＋B＞C、A＋C＞B、B＋C＞A，才能构成三角形。但如果把六个不等式中的任何一个大于号“＞”错写成大于等于号“≥”，那就不能构成三角形。问题恰出现在容易被疏忽的边界附近。这里所说

的边界是指，相当于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。

使用边界值分析方法设计测试用例，首先应确定边界情况。通常输入等价类与输出等价类的边界，就是应着重测试的边界情况。应当选取正好等于，刚刚大于，或刚刚小于边界的值做为测试数据，而不是选取等价类中的典型值或任意值做为测试数据。

边界值分析方法是最有效的黑盒测试方法，但当边界情况很复杂的时候，要找出适当的测试用例还需针对问题的输入域、输出域边界，耐心细致地逐个考虑。

(7) 错误推测法

人们也可以靠经验和直觉推测程序中可能存在的各种错误，从而有针对性地编写检查这些错误的例子。这就是错误推测法。

错误推测法的基本想法是：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。例如，在介绍单元测试时曾列出许多在模块中常见的错误，这些是单元测试经验的总结。此外，对于在程序中容易出错的情况，也有一些经验总结出来。例如，输入数据为 0，或输出数据为 0 是容易发生错误的情形，因此可选择输入数据为 0，或使输出数据为 0 的例子作为测试用例。又例如，输入表格为空或输入表格只有一行，也是容易发生错误的情况。可选择表示这种情况的例子作为测试用例。再例如，可以针对一个排序程序，输入空的值（没有数据）、输入一个数据、让所有的输入数据都相等、让所有输入数据有序排列、让所有输入数据逆序排列等，进行错误推测。

(8) 因果图

前面介绍的等价类划分方法和边界值分析方法，都是着重考虑输入条件，但未考虑输入条件之间的联系。如果在测试时必须考虑输入条件的各种组合，可能的组合数将是天文数字。因此必须考虑使用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来考虑设计测试用例，这就需要利用因果图。 因果图方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况。

利用因果图生成测试用例的基本步骤是：

§ 分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。

§ 分析软件规格说明描述中的语义，找出原因与结果之间，原因与原因之间对应的是什么关系？根据这些关系，画出因果图。

§ 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

§ 把因果图转换成判定表。

§ 把判定表的每一列拿出来作为依据，设计测试用例。

通常，在因果图中，用 Ci 表示原因，Ei 表示结果，其基本符号如图 5.15 所示。各结点表示状态，可取值“0”或“1”。“0”表示某状态不出现，“1”表示某状态出现。

§ 恒等：若原因出现，则结果出现。若原因不出现，则结果也不出现。

§ 非：若原因出现，则结果不出现。若原因不出现，反而结果出现。

§ 或（∨）：若几个原因中有一个出现，则结果出现，几个原因都不出现，结果不出现。

§ 与（∧）：若几个原因都出现，结果才出现。若其中有一个原因不出现，结果不出现。

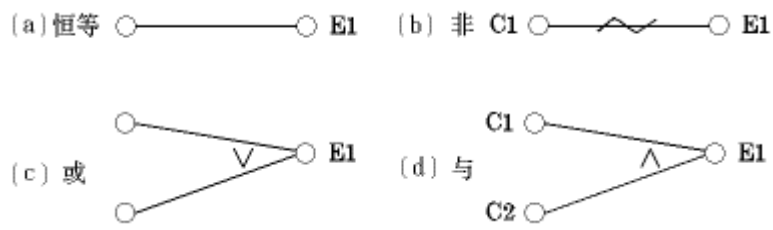


图 5.15 因果图的图形符号

为了表示原因与原因之间，结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。从输入（原因）考虑，有四种约束；从输出（结果）考虑，还有一种约束，参看图 5.16：

§ E（互斥）：表示 a，b 两个原因不会同时成立，两个中最多有一个可能成立。
§ I（包含）：表示 a，b，c 三个原因中至少有一个必须成立。
§ O（唯一）：表示 a 和 b 当中必须有一个，且仅有一个成立。
§ R（要求）：表示当 a 出现时，b 必须也出现。不可能 a 出现，b 不出现。

§ M（屏蔽）：表示当 a 是 1 时，b 必须是 0。而当 a 为 0 时，b 的值不定。

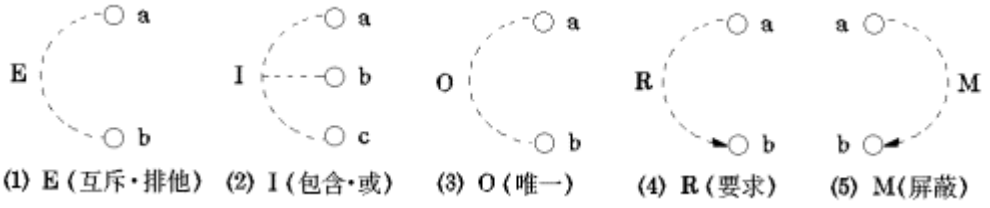


图 5.16 因果图的约束符号

【例】有一个处理单价为 5 角钱的饮料的自动售货机软件测试用例的设计。 其规格说明如下：“若投入 5 角钱或 1 元钱的硬币，押下〖橙汁〗或〖啤酒〗的按钮，则相应的饮料就送出来。若售货机没有零钱找，则一个显示〖零钱找完〗的红灯亮，这时在投入 1 元硬币并押下按钮后，饮料不送出来而且 1 元硬币也退出来；若有零钱找，则显示〖零钱找完〗的红灯灭，在送出饮料的同时退还 5 角硬币。”

§ 分析这一段说明，列出原因和结果

原因：
1.售货机有零钱找
2.投入 1 元硬币
3.投入 5 角硬币
4.押下橙汁按钮
5.押下啤酒按钮
结果：
21. 售货机〖零钱找完〗灯亮
22. 退还 1 元硬币
23. 退还 5 角硬币
24. 送出橙汁饮料
25. 送出啤酒饮料

§ 画出因果图，如图 5.17 所示。所有原因结点列在左边，所有结果结点列在右边。

建立两个中间结点，表示处理的中间状态。

中间结点：
11. 投入 1 元硬币且押下饮料按钮

12. 押下〖橙汁〗或〖啤酒〗的按钮

13. 应当找 5 角零钱并且售货机有零钱找

14. 钱已付清

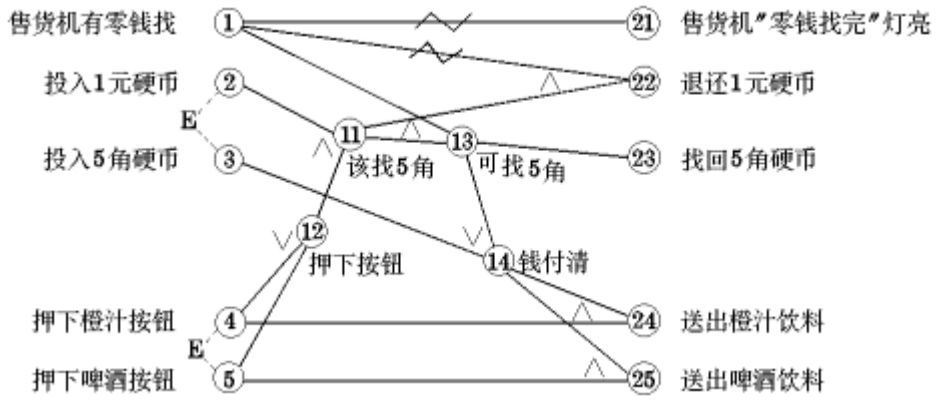


图 5.17 因果图

§ 由于 2 与 3 ，4 与 5 不能同时发生，分别加上约束条件 E。
§ 转换成判定表：

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

图 5.18 由因果图得到的判定表

在判定表中，阴影部分表示因违反约束条件的不可能出现的情况，删去。第 16 列与第 32 列因什么动作也没做，也删去。最后可根据剩下的 16 列作为确定测试用例的依据。

因果图方法是一个非常有效的黑盒测试方法，它能够生成没有重复性的且发现错误能力强的测试用例，而且对输入、输出同时进行了分析。 (9) 测试方法选择的综合策略

Myers 提出了使用各种测试方法的综合策略:

§ 在任何情况下都必须使用边界值分析方法。经验表明用这种方法设计出测试用例发现程序错误的能力最强。

§ 必要时用等价类划分方法补充一些测试用例。

§ 用错误推测法再追加一些测试用例。

§ 对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度。如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

§ 如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

5. 程序的静态测试

(1) 源程序静态分析

通常采用以下一些方法进行源程序的静态分析。

① 生成各种引用表

§ 直接从表中查出说明 / 使用错误等。如，循环层次表、变量交叉引用表、标号交叉引用表等。

§ 为用户提供辅助信息。如，子程序（宏、函数）引用表、等价（变量、标号）表、常数表等。

§ 用来做错误预测和程序复杂度计算。如，操作符和操作数的统计表等。

② 静态错误分析

静态错误分析主要用于确定在源程序中是否有某类错误或“危险”结构。

§ 类型和单位分析 ：为了强化对源程序中数据类型的检查，发现在数据类型上的错误和单位上的不一致性，在程序设计语言中扩充了一些结构。如单位分析要求使用一种预处理器，它能够通过使用一般的组合 / 消去规则，确定表达式的单位。

§ 引用分析 ：最广泛使用的静态错误分析方法就是发现引用异常。如果沿着程序的控制路径，变量在赋值以前被引用，或变量在赋值以后未被引用，这时就发生了引用异常。为了检测引用异常，需要检查通过程序的每一条路径。也可以建立引用异常的探测工具。

§ 表达式分析 ：对表达式进行分析，以发现和纠正在表达式中出现的错误。包括：在表达式中不正确地使用了括号造成错误。数组下标越界造成错误。除式为

零造成错误。对负数开平方，或对 π 求正切值造成错误。以及对浮点数计算的误差进行检查。

§ 接口分析 ：关于接口的静态错误分析主要检查过程、函数过程之间接口的一致性。因此要检查形参与实参在类型、数量、维数、顺序、使用上的一致性；检查全局变量和公共数据区在使用上的一致性。

(2) 人工测试

静态分析中进行人工测试的主要方法有桌前检查、代码审查和走查。经验表明，使用这种方法能够有效地发现 30％到 70％的逻辑设计和编码错误。

① 桌前检查（Desk Checking）

由程序员自己检查自己编写的程序。程序员在程序通过编译之后，进行单元测试设计之前，对源程序代码进行分析，检验，并补充相关的文档，目的是发现程序中的错误。检查项目有：

§ 检查变量的交叉引用表 ：重点是检查未说明的变量和违反了类型规定的变量；还要对照源程序，逐个检查变量的引用、变量的使用序列；临时变量在某条路径上的重写情况；局部变量、全局变量与特权变量的使用；

§ 检查标号的交叉引用表 ：验证所有标号的正确性：检查所有标号的命名是否正确；转向指定位置的标号是否正确。

§ 检查子程序、宏、函数 ：验证每次调用与被调用位置是否正确；确认每次被调用的子程序、宏、函数是否存在；检验调用序列中调用方式与参数顺序、个数、类型上的一致性。

§ 等值性检查 ：检查全部等价变量的类型的一致性，解释所包含的类型差异。

§ 常量检查 ：确认每个常量的取值和数制、数据类型；检查常量每次引用同它的取值、数制和类型的一致性；

§ 标准检查 ：用标准检查程序或手工检查程序中违反标准的问题。

§ 风格检查 ：检查在程序设计风格方面发现的问题。

§ 比较控制流 ：比较由程序员设计的控制流图和由实际程序生成的控制流图，寻找和解释每个差异，修改文档和校正错误。

§ 选择、激活路径 ：在程序员设计的控制流图上选择路径，再到实际的控制流图上激活这条路径。如果选择的路径在实际控制流图上不能激活，则源程序可能有错。用这种方法激活的路径集合应保证源程序模块的每行代码都被检查，即桌前检查应至少是语句覆盖。

§ 对照程序的规格说明，详细阅读源代码 ：程序员对照程序的规格说明书、规定的算法和程序设计语言的语法规则，仔细地阅读源代码，逐字逐句进行分析和思考，比较实际的代码和期望的代码，从它们的差异中发现程序的问题和错误。

§ 补充文档 ：桌前检查的文档是一种过渡性的文档，不是公开的正式文档。通过编写文档，也是对程序的一种下意识的检查和测试，可以帮助程序员发现和抓住更多的错误。

这种桌前检查，由于程序员熟悉自己的程序和自身的程序设计风格，可以节省很多的检查时间，但应避免主观片面性。

② 代码会审（Code Reading Review）

代码会审是由若干程序员和测试员组成一个会审小组，通过阅读、讨论和争议，对程序进行静态分析的过程。

代码会审分两步：第一步，小组负责人提前把设计规格说明书、控制流程图、程序文本及有关要求、规范等分发给小组成员，作为评审的依据。小组成员在充分阅读这些材料之后，进入审查的第二步：召开程序审查会。在会上，首先由程序员逐句讲解程序的逻辑。在此过程中，程序员或其他小组成员可以提出问题，展开讨论，审查错误是否存在。实践表明，程序员在讲解过程中能发现许多原来自己没有发现的错误，而讨论和争议则促进了问题的暴露。

在会前，应当给会审小组每个成员准备一份常见错误的清单，把以往所有可能发生的常见错误罗列出来，供与会者对照检查，以提高会审的实效。这个常见错误清单也叫做检查表，它把程序中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误，然后把它们制成表格，供在会审时使用。这种检查表类似于本章单元测试中给出的检查表。

③ 走查（Walkthroughs）

走查与代码会审基本相同，其过程分为两步。第一步也把材料先发给走查小组每个成员，让他们认真研究程序，然后再开会。开会的程序与代码会审不同，不是简单地读程序和对照错误检查表进行检查，而是让与会者“充当”计算机。即首先由测试组成员为被测程序准备一批有代表性的测试用例，提交给走查小组。走查小组开会，集体扮演计算机角色，让测试用例沿程序的逻辑运行一遍，随时记录程序的踪迹，供分析和讨论用。 人们借助于测试用例的媒介作用，对程序的逻辑和功能提出各种疑问，结合问题开展热烈的讨论和争议，能够发现更多的问题。

6. 调试（Debug，排错）

软件测试也是一个系统工程，在做测试时，需要先做测试计划和规格说明，然后设计测试用例，定义策略，最后将测试结果与预先给出的期望结果进行比较，再做评价分析。而软件调试则是在进行了成功的测试之后才开始的工作。它与软件

测试不同，软件测试的目的是尽可能多地发现软件中的错误，但进一步诊断和改正程序中潜在的错误，则是调试的任务。

调试活动由两部分组成：

① 确定程序中可疑错误的确切性质和位置。

② 对程序（设计，编码）进行修改，排除这个错误。

通常，调试工作是一个具有很强技巧性的工作。一个软件工程人员在分析测试结果的时候会发现，软件运行失效或出现问题，往往只是潜在错误的外部表现，而外部表现与内在原因之间常常没有明显的联系。如果要找出真正的原因，排除潜在的错误，不是一件易事。因此可以说，调试是通过现象，找出原因的一个思维分析的过程。

(1) 调试的步骤

① 从错误的外部表现形式入手，确定程序中出错位置；

② 研究有关部分的程序，找出错误的内在原因；

③ 修改设计和代码，以排除这个错误；

④ 重复进行暴露了这个错误的原始测试或某些有关测试，以确认该错误是否被排除；是否引进了新的错误。

⑤ 如果所做的修正无效，则撤消这次改动，重复上述过程，直到找到一个有效的解决办法为止。

从技术角度来看，查找错误的难度在于：

§ 现象与原因所处的位置可能相距甚远。就是说，现象可能出现在程序的一个部位，而原因可能在离此很远的另一个位置。高耦合的程序结构中这种情况更为明显。

§ 当纠正其它错误时，这一错误所表现出的现象可能会暂时消失，但并未实际排除。

§ 现象实际上是由一些非错误原因（例如，舍入得不精确）引起的。

§ 现象可能是由于一些不容易发现的人为错误引起的。

§ 错误是由于时序问题引起的，与处理过程无关。

§ 现象是由于难于精确再现的输入状态（例如，实时应用中输入顺序不确定）引起。

§ 现象可能是周期出现的。在软、硬件结合的嵌入式系统中常常遇到。

(2) 几种主要的调试方法

调试的关键在于推断程序内部的错误位置及原因。为此，可以采用以下方法：

① 强行排错

这是目前使用较多，效率较低的调试方法。它不需要过多的思考，比较省脑筋。例如：

§ 通过内存全部打印来排错（Memory Dump）；

§ 在程序特定部位设置打印语句；

§ 自动调试工具。

可供利用的典型的语言功能有：打印出语句执行的追踪信息，追踪子程序调用，以及指定变量的变化情况。自动调试工具的功能是：设置断点，当程序执行到某个特定的语句或某个特定的变量值改变时，程序暂停执行。程序员可在终端上观察程序此时的状态。

应用以上任一种方法之前，都应当对错误的征兆进行全面彻底的分析，得出对出错位置及错误性质的推测，再使用一种适当的排错方法来检验推测的正确性。

② 回溯法排错

这是在小程序中常用的一种有效的排错方法。一旦发现了错误，人们先分析错误征兆，确定最先发现“症状”的位置。然后，人工沿程序的控制流程，向回追踪源程序代码，直到找到错误根源或确定错误产生的范围。

回溯法对于小程序很有效，往往能把错误范围缩小到程序中的一小段代码；仔细分析这段代码不难确定出错的准确位置。但对于大程序，由于回溯的路径数目较多，回溯会变得很困难。

③ 归纳法排错

归纳法是一种从特殊推断一般的系统化思考方法。归纳法排错的基本思想是：从一些线索（错误征兆）着手，通过分析它们之间的关系来找出错误。

归纳法排错步骤大致分为以下四步：

§ 收集有关的数据 ：列出所有已知的测试用例和程序执行结果。看哪些输入数据的运行结果是正确的，哪些输入数据的运行结果有错误存在。

§ 组织数据 ：由于归纳法是从特殊到一般的推断过程，所以需要组织整理数据，以便发现规律。常用的构造线索的技术是“分类法”。

	Yes	No
What（列出一般现象）		
Where（说明发现现象的地点）		
When（列出现象发生时所有已知情况）		
How（说明现象的范围和量级）		

而在“Yes”和“No”这两列中，“Yes”描述了出现错误的现象的 3W1H，“No”作为比较，描述了没有错误的现象的 3W1H。通过分析，找出矛盾来。

§ 提出假设 ：分析线索之间的关系，利用在线索结构中观察到的矛盾现象，设计一个或多个关于出错原因的假设。如果一个假设也提不出来，归纳过程就需要收集更多的数据。此时，应当再设计与执行一些测试用例，以获得更多的数据。如果提出了许多假设，则首先选用最有可能成为出错原因的假设。

§ 证明假设 ：把假设与原始线索或数据进行比较，若它能完全解释一切现象，则假设得到证明；否则，就认为假设不合理，或不完全，或是存在多个错误，以致只能消除部分错误。

④ 演绎法排错

演绎法是一种从一般原理或前提出发，经过排除和精化的过程来推导出结论的思考方法。演绎法排错是测试人员首先根据已有的测试用例，设想及枚举出所有可能出错的原因做为假设；然后再用原始测试数据或新的测试，从中逐个排除不可能正确的假设；最后，再用测试数据验证余下的假设确是出错的原因。

演绎法主要有以下四个步骤：

§ 列举所有可能出错原因的假设 ：把所有可能的错误原因列成表。它们不需要完全的解释，而仅仅是一些可能因素的假设。通过它们，可以组织、分析现有数据。

§ 利用已有的测试数据，排除不正确的假设 ：仔细分析已有的数据，寻找矛盾，力求排除前一步列出所有原因。如果所有原因都被排除了，则需要补充一些数据（测试用例），以建立新的假设；如果保留下来的假设多于一个，则选择可能性最大的原因做基本的假设。

§ 改进余下的假设 ：利用已知的线索，进一步改进余下的假设，使之更具体化，以便可以精确地确定出错位置。

§ 证明余下的假设 ：这一步极端重要，具体做法与归纳法的第 4 步相同。

(3) 调试原则

在调试方面，许多原则本质上是心理学方面的问题。因为调试由两部分组成，所以调试原则也分成两组。

① 确定错误的性质和位置的原则

§ 用头脑去分析思考与错误征兆有关的信息。最有效的调试方法是用头脑分析与错误征兆有关的信息。一个能干的程序调试员应能做到不使用计算机就能够确定大部分错误。

§ 避开死胡同。如果程序调试员走进了死胡同，或者陷入了绝境，最好暂时把问题抛开，留到第二天再去考虑，或者向其他人讲解这个问题。事实上常有这种情形：向一个好的听众简单地描述这个问题时，不需要任何听讲者的提示，你自己会突然发现问题的所在。

§ 只把调试工具当做辅助手段来使用。利用调试工具，可以帮助思考，但不能代替思考。因为调试工具给你的是一种无规律的调试方法。实验证明，即使是对一个不熟悉的程序进行调试时，不用工具的人往往比使用工具的人更容易成功。

§ 避免用试探法，最多只能把它当做最后手段。初学调试的人最常犯的一个错误是想试试修改程序来解决问题。这还是一种碰运气的盲目的动作，它的成功机会很小，而且还常把新的错误带到问题中来。

② 修改错误的原则

§ 在出现错误的地方，很可能还有别的错误。经验证明，错误有群集现象，当在某一程序段发现有错误时，在该程序段中还存在别的错误的概率也很高。因此，在修改一个错误时，还要查一下它的近邻，看是否还有别的错误。

§ 修改错误的一个常见失误是只修改了这个错误的征兆或这个错误的表现，而没有修改错误的本身。如果提出的修改不能解释与这个错误有关的全部线索，那就表明了只修改了错误的一部分。

§ 当心修正一个错误的同时有可能会引入新的错误。人们不仅需要注意不正确的修改，而且还要注意看起来是正确的修改可能会带来的副作用，即引进新的错误。因此在修改了错误之后，必须进行回归测试，以确认是否引进了新的错误。

§ 修改错误的过程将迫使人们暂时回到程序设计阶段。修改错误也是程序设计的一种形式。一般说来，在程序设计阶段所使用的任何方法都可以应用到错误修正的过程中来。

§ 修改源代码程序，不要改变目标代码。

7. 测试中的可靠性分析

在软件开发的过程中，利用测试的统计数据，估算软件的可靠性，以控制软件的质量是至关重要的。

(1) 推测错误的产生频度

估算错误产生频度的一种方法是估算平均失效等待时间 MTTF（Mean Time To Failure）。MTTF 估算公式（Shooman 模型）是

$$MTTF = \frac{1}{K(E_T/I_T - E_c(t)/I_T)}$$

其中，K 是一个经验常数，美国一些统计数字表明，K 的典型值是 200；

ET 是测试之前程序中原有的故障总数；

IT 是程序长度（机器指令条数或简单汇编语句条数）；

t 是测试（包括排错）的时间；

EC（t）是在 0～t 期间内检出并排除的故障总数。
公式的基本假定是：

§ 单位(程序)长度中的故障数 ET／IT 近似为常数，它不因测试与排错而改变。统计数字表明，通常 ET／IT 值的变化范围在 0.5×10-2～2×10-2 之间；

§ 故障检出率正比于程序中残留故障数，而 MTTF 与程序中残留故障数成正比；

§ 故障不可能完全检出，但一经检出立即得到改正。

下面对此问题做一分析：

设 EC（τ）是 0～τ 时间内检出并排除的故障总数，τ 是测试时间(月)，则在同一段时间 0～τ 内的单条指令累积规范化排除故障数曲线 εc（τ）为：
εc（τ）= EC（τ）／IT

这条曲线在开始呈递增趋势，然后逐渐和缓，最后趋近于一水平的渐近线 ET／IT。利用公式的基本假定：故障检出率（排错率）正比于程序中残留故障数及残留故障数必须大于零，经过推导得：

$$\varepsilon_c(t) = \frac{E_T}{I_T}\left(1 - e^{-K_1\tau}\right)$$

这就是故障累积的 S 型曲线模型，参看图 5.19。

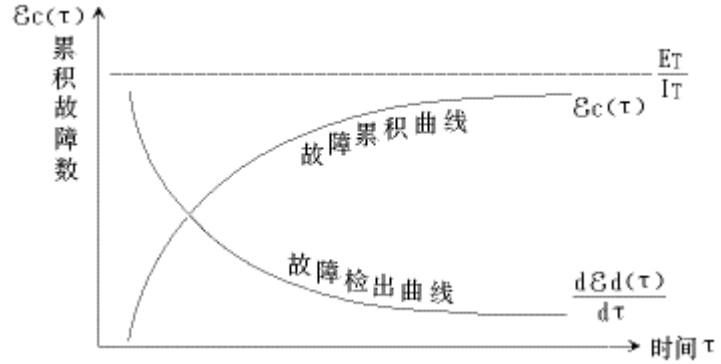


图 5.19 故障累积曲线与故障检出曲线

故障检出曲线服从指数分布，亦在图 5.19 中显示。

$$\frac{d\varepsilon_c(t)}{d\tau}=K_1\left(\frac{E_T}{I_T}e^{-K_1\tau}\right)$$

(2) 估算软件中故障总数 ET 的方法

①利用 Shooman 模型估算程序中原来错误总量 ET —瞬间估算

$$MTTF=\frac{1}{K(E_T/I_T-E_c(t)/I_T)}=\frac{1}{\lambda}$$

所以，

$$\lambda=K\left(\frac{E_T}{I_T}-\frac{E_c(t)}{I_T}\right)$$

若设 T 是软件总的运行时间，M 是软件在这段时间内的故障次数，则

T / M = 1 / λ = MTTF

现在对程序进行两次不同的互相独立的功能测试，相应检错时间 τ1 < τ2，检出的错误数 EC（τ1）< EC（τ2），则有

$$\lambda_1=\frac{E_c(\tau_1)}{\tau_1}=\frac{1}{MTTF_1}$$

$$\lambda_2=\frac{E_c(\tau_2)}{\tau_2}=\frac{1}{MTTF_2}$$

且

$$\lambda_1 = K\left(\frac{E_I}{I_I} - \frac{E_c(\tau_1)}{I_I}\right)$$

$$\lambda_2 = K\left(\frac{E_I}{I_I} - \frac{E_c(\tau_2)}{I_I}\right)$$

解上述方程组，得到 ET 的估计值和 K 的估计值。

$$\hat{E}_I = \frac{E_c(\tau_2)\lambda_1 - E_c(\tau_1)\lambda_2}{\lambda_1 - \lambda_2}$$

$$\hat{K} = \frac{I_I\lambda_1}{E_I - E_c(\tau_1)} \quad 或\ \hat{K} = \frac{I_I\lambda_2}{E_I - E_c(\tau_2)}$$

② 利用植入故障法估算程序中原有故障总数 ET ― 捕获－再捕获抽样法

若设 NS 是在测试前人为地向程序中植入的故障数（称播种故障），nS 是经过一段时间测试后发现的播种故障的数目，n0 是在测试中又发现的程序原有故障数。设测试用例发现植入故障和原有故障的能力相同，则程序中原有故障总数 ET 的估算值为

$$\hat{N}_o = \frac{n_o * N_s}{n_s}$$

在此方法中要求对播种故障和原有故障同等对待，因此可以由对这些植入的已知故障一无所知的测试专业小组进行测试。

这种对播种故障的捕获—再捕获的抽样方法显然需要消耗许多时间在发现和修改播种故障上，这会影响工程的进度，而且要想使植入的故障有利于精确地推测原有的故障数，如何选择和植入这些播种故障也是一件很困难的事情。为了回避这些难点，就有了下面不必埋设播种故障的方法。

③ Hyman 分别测试法

这是对植入故障法的一种补充。由两个测试员同时互相独立地测试同一程序的两个副本，用 t 表示测试时间（月），记 t = 0 时，程序中原有故障总数是 B0；t

= t1 时，测试员甲发现的故障总数是 B1；测试员乙发现的故障总数是 B2；其中两人发现的相同故障数目是 bc；两人发现的不同故障数目是 bi。

在大程序测试时，头几个月所发现的错误在总的错误中具有代表性，两个测试员测试的结果应当比较接近，bi 不是很大。这时有

$$B_0 = \frac{B_1 * B_2}{b_c}$$

如果 bi 比较显著，应当每隔一段时间，由两个测试员再进行分别测试，分析测试结果，估算 B0。如果 bi 减小，或几次估算值的结果相差不多，则可用 B0 作为程序中原有错误总数 ET 的估算值。

三、例题分析

【例 1】为了把握软件开发各个环节的正确性和协调性，人们需要进行（ A ）和（ B ）工作。（ A ）的目的是想证实在一给定的外部环境中软件的逻辑正确性。它包括（ C ）和（ D ），（ B ）则试图证明在软件生存期各个阶段，以及阶段间的逻辑（ E ）、（ F ）和正确性。

供选择的答案：

A, B. ① 操作 ② 确认 ③ 验证 ④ 测试 ⑤ 调试

C, D. ① 用户的确认 ② 需求规格说明的确认

③ 程序的确认 ④ 测试的确认

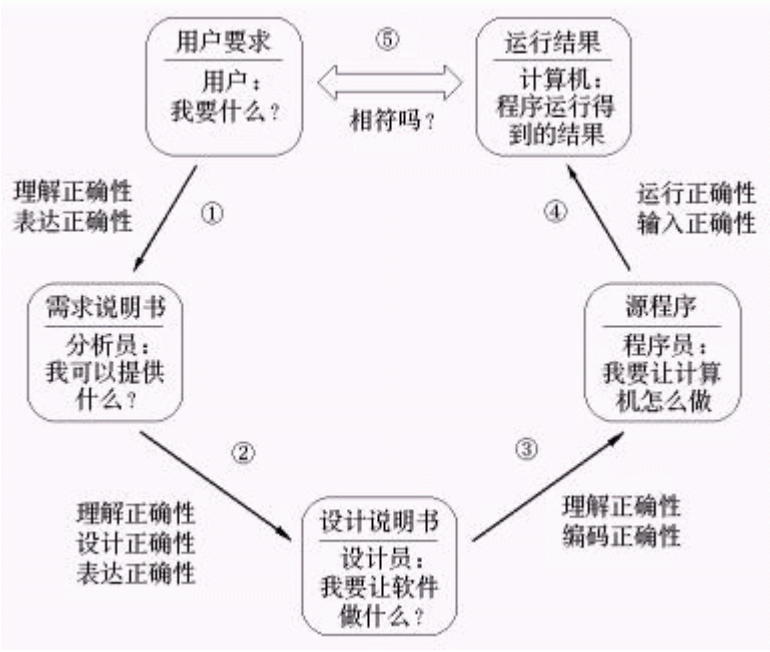
E, F. ① 可靠性 ② 独立性 ③ 协调性 ④ 完备性 ⑤ 扩充性

答案：A. ② B. ③ C. ② D. ③ E. ③ F. ④

分析：到程序的测试为止，软件开发工作已经经历了许多环节，每个环节都可能发生问题。为了把握各个环节的正确性，人们需要进行各种确认和验证工作。

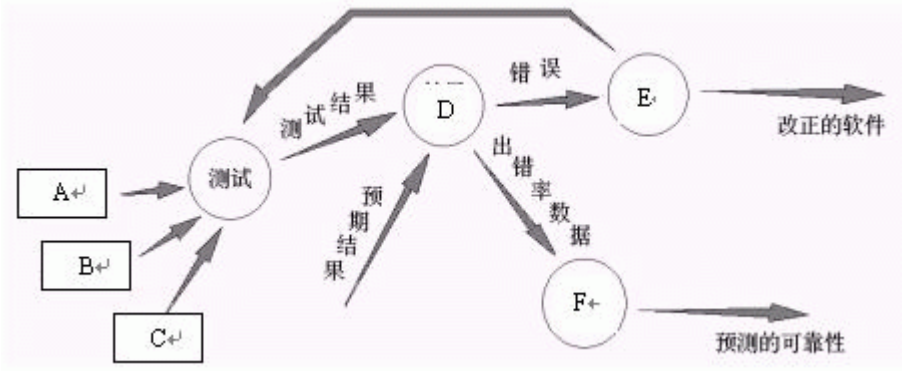
所谓确认，是一系列的活动和过程，其目的是想证实在一个给定的外部环境中软件的逻辑正确性。它包括需求规格说明的确认和程序的确认，而程序的确认又分为静态确认与动态确认。静态确认一般不在计算机上实际执行程序，而是通过人工分析或者程序正确性证明来确认程序的正确性;动态确认主要通过动态分析和程序测试来检查程序的执行状态，以确认程序是否有问题。

所谓验证，则试图证明在软件生存期各个阶段，以及阶段间的逻辑协调性、完备性和正确性。下图中所示的就是软件生存期各个重要阶段之间所要保持的正确性。它们就是验证工作的主要对象。



确认与验证工作都属于软件测试。在对需求理解与表达的正确性、设计与表达的正确性、实现的正确性以及运行的正确性的验证中，任何一个环节上发生了问题都可能在软件测试中表现出来。

【例 2】测试过程需要三类输入：（ A ）、（ B ）和（ C ）。请选择正确的答案填入下图中以完成测试信息处理的全过程。



供选择的答案：

A ~ C. ① 接口选择 ② 软件配置 ③ 硬件配置

④ 测试配置 ⑤ 测试环境 ⑥ 测试工具

D ~ F. ① 排错 ② 可靠性分析 ③ 结果分析 ④ 数据分类

答案：A. ② B. ④ C. ⑥ D. ③ E. ① F. ②

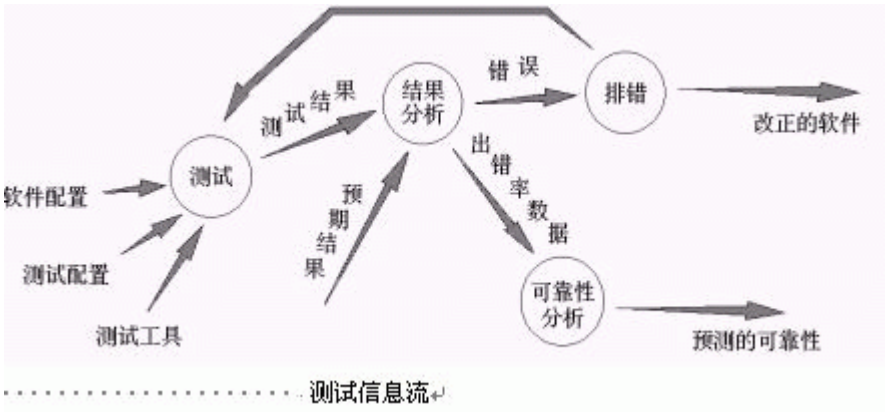
分析：测试信息流如图所示。测试过程需要三类输入：

（1）软件配置：包括软件需求规格说明、软件设计规格说明、源代码等；

（2）测试配置：包括表明测试工作如何进行的测试计划、给出测试数据的测试用例、控制测试进行的测试程序等；实际上，测试配置是软件配置的一个子集。

（3）测试工具：为提高软件测试效率，测试工作需要有测试工具的支持，它们的工作就是为测试的实施提供某种服务。例如，测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序、以及驱动测试的测试数据库等等。

测试之后，要对所有测试结果进行分析，即将实测的结果与预期的结果进行比较。如果发现出错的数据，就意味着软件有错误，然后就需要开始排错（调试）。即对已经发现的错误进行错误定位和确定出错性质，并改正这些错误，同时修改相关的文档。修正后的文档一般都要经过再次测试，直到通过测试为止。通过收集和分析测试结果数据，开始对软件建立可靠性模型。



最后，如果测试发现不了错误，那么几乎可以肯定，测试配置考虑得不够细致充分，错误仍然潜伏在软件中。这些错误最终不得不由用户在使用中发现，并在维护时由开发者去改正。但那时改正错误的费用将比在开发阶段改正错误的费用要高出 40 倍到 60 倍。

【例 3】软件测试是软件质量保证的主要手段之一，测试的费用已超过（ A ）的 30%以上。因此，提高测试的有效性十分重要。“高产”的测试是指（ B ）。根据国家标准 GB 8566－88《计算机软件开发规范》的规定，软件的开发和维护划分为 8 个阶段，其中，单元测试是在（ C ）阶段完成的，集成测试的计划是在（ D ）阶段制定的，确认测试的计划是在（ E ）阶段制定的。

供选择的答案：

A. ① 软件开发费用 ② 软件维护费用 ③ 软件开发和维护费用

④ 软件研制费用 ⑤ 软件生存期全部

B. ① 用适量的测试用例运行程序，证明被测程序正确无误

② 用适量的测试用例运行程序，证明被测程序符合相应的要求

③ 用少量的测试用例运行程序，发现被测程序尽可能多的错误

④ 用少量的测试用例运行程序，纠正被测程序尽可能多的错误

C ~ E. ① 可行性研究和计划 ② 需求分析 ③ 概要设计

④ 详细设计 ⑤ 实现 ⑥ 集成测试

⑦ 确认测试 ⑧ 使用和维护
答案：A. ① B. ③ C. ⑤ D. ③ E. ②

分析：由于原始问题的复杂性，软件本身的复杂性和抽象性，软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。这些错误和缺陷如果在软件交付投入生产性运行之前不能加以排除的话，在运行中迟早会暴露出来。但到那时，不仅改正这些错误的代价更高，而且往往造成很恶劣的后果。因此，为了保证所开发软件的质量，测试是不可少的。测试的任务就是设计出测试用例，运行被测程序，发现软件中隐藏的的各种错误。目前在大中型软件项目的开发中。测试占据着重要的地位，测试所花费费用已超过软件开发费用的 70%以上。

如何组织好测试，特别是如何选择好的测试用例，对于降低测试费用，保障测试质量有着重要的意义。一个高效的、或“高产”的测试，是指设计少量的测试用例，用它们来运行程序，发现被测程序的尽可能多的问题。 测试按照测试的内容可分为三种：单元测试（对程序单元或模块单独进行测试）、集成测试（把已通过单元测试的模块按照设计的要求组装起来，测试模块间的接口以及设计中的问题）和确认测试（对软件的有效性做全面测试，以确定开发的软件是否符合验收标准）。

根据国标 GB 8566－88《计算机软件开发规范》的规定，软件的开发和维护划分为 8 个阶段：可行性研究及计划、需求分析、概要设计、详细设计、实现、集成测试、确认测试、使用维护，并且规定，单元测试是在实现阶段完成的，集成测试的计划应当在概要设计阶段制定的，确认测试的计划应当在需求分析阶段制定的。

【例 4】集成测试也叫做（ A ）或（ B ）。通常，在（ C ）的基础上，将所有模块按照设计要求组装成为系统。子系统的集成测试特别称为（ D ），它所做的工作是要找出子系统和系统需求规格说明之间的（ E ）。需要考虑的问题是：在把各个模块连接起来的时侯，穿越模块接口的数据是否会（ F ）； 一个模块的功能是否会对另一个模块的功能产生不利的影响；各个（ G ）组合起来，能否达到预期要求的（ H ）；（ I ）是否有问题；单个模块的误差累积起来是否会放大。

供选择的答案：

A ~ D. ① 单元测试 ② 部件测试 ③ 组装测试

④ 系统测试 ⑤ 确认测试 ⑥ 联合测试

E ~ I. ① 子功能 ② 丢失 ③ 父功能 ④ 局部数据结构

⑤ 全局数据结构 ⑥ 不一致 ⑦ 一致
答案：A. ③ B. ⑥ C. ① D. ② E. ⑥ F. ② G. ① H. ③ I. ⑤

A、B 的答案可互换

分析：集成测试也叫做组装测试或联合测试。通常，在单元测试的基础上，将所有模块按照设计要求组装成为系统。子系统的集成测试特别称为部件测试，它所做的工作是要找出子系统和系统需求规格说明之间的不一致。这时需要考虑的问题是：

(1) 在把各个模块连接起来的时侯，穿越模块接口的数据是否会丢失；

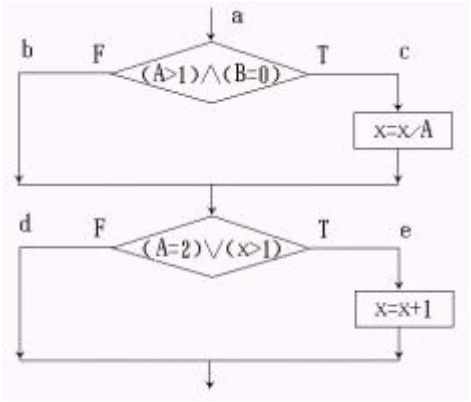
(2) 一个模块的功能是否会对另一个模块的功能产生不利的影响；

(3) 各个子功能组合起来，能否达到预期要求的父功能；

(4) 全局数据结构是否有问题；

(5) 单个模块的误差累积起来是否会放大。

【例 5】如图所示的程序有四条不同的路径。分别表示为 L1(a→c→e)、L2(a→b→d)、L3(a→b→e) 和 L4(a→c→d)，或简写为 ace、abd、abe 及 acd。由于覆盖测试的目标不同，逻辑覆盖方法可以分为语句覆盖、判定覆盖、条件覆盖、判定－条件覆盖、条件组合覆盖和路径覆盖。



从备选的答案中选择适当的测试用例与之匹配。（ A ）属于语句覆盖；（ B ）、（ C ）属于判定覆盖；（ D ）、（ E ）属于条件覆盖；（ F ）、（ G ）属于判定－条件覆盖；（ H ）属于条件组合覆盖；（ I ）属于路径覆盖。

供选择的答案：

A. ~ I. ① 【(2,0,4)，(2,0,3)】覆盖 ace；

【(1,1,1)，(1,1,1)】覆盖 abd；

判断条件覆盖

② 【(1,0,3)，(1,0,4)】覆盖 abe；

【(2,1,1)，(2,1,2)】覆盖 abe；条件覆盖

③ 【(2,0,4)，(2,0,3)】覆盖 ace；

④ 【(2,1,1)，(2,1,2)】覆盖 abe； 【(3,0,3)，(3,1,1)】覆盖 acd；

⑤ 【(2,0,4)，(2,0,3)】覆盖 ace； 【(1,0,1)，(1,0,1)】覆盖 abd；

【(2,1,1)，(2,1,2)】覆盖 abe； 条件覆盖， 判条

⑥ 【(2,0,4)，(2,0,3)】覆盖 ace 【(1,1,1)，(1,1,1)】覆盖 abd；路径覆盖

【(1,1,2)，(1,1,3)】覆盖 abe； 【(3,0,3)，(3,0,1)】覆盖 acd；

⑦ 【(2,0,4)，(2,0,3)】覆盖 ace； 【(1,1,1)，(1,1,1)】覆盖 abd； 条件组合

【(1,0,3)，(1,0,4)】覆盖 abe； 【(2,1,1)，(2,1,2)】覆盖 abe；

答案：A. ③ B. ① C. ④ D. ② E. ⑤ F. ① G. ⑤ H. ⑦ I. ⑥

B、C 的答案可互换。D、E 的答案可互换。F、G 的答案可互换。

分析：(1) 语句覆盖

所谓语句覆盖就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。例如在本题所给出的例子中，正好所有的可执行语句都在路径 L1 上， 所以选择路径 L1 设计测试用例，就可以覆盖所有的可执行语句。 测试用例的设计格式如下
【输入的（A, B, x），输出的（A, B, x）】

本题满足语句覆盖的测试用例是：
【(2, 0, 4)，(2, 0, 3)】 覆盖 ace【L1】 （答案③）

从程序中每个可执行语句都得到执行这一点来看，语句覆盖的方法似乎能够比较全面地检验每一个可执行语句。但需要注意的是，这种覆盖也绝不是完美无缺的。假设图中两个判断的逻辑运算有问题，例如，第一个判断中的逻辑运算符“∧”错写成了“∨”，或者第二个判断中的逻辑运算符“∨”错写成了“∧”，利用上面的测试用例，仍可覆盖所有 4 个可执行语句。这说明虽然做到了语句覆盖，

但可能发现不了判断中逻辑运算中出现的错误。与后面所介绍的其它覆盖相比，语句覆盖是最弱的逻辑覆盖准则。

(2) 判定覆盖

所谓判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。判定覆盖又称为分支覆盖。对于本题所给出的例子，如果选择路径 L1 和 L2，就可得满足要求的测试用例：
【(2, 0, 4)，(2, 0, 3)】 覆盖 ace【L1】 （答案①）
【(1, 1, 1)，(1, 1, 1)】 覆盖 abd【L2】
如果选择路径 L3 和 L4，还可得另一组可用的测试用例:
【(2, 1, 1)，(2, 1, 2)】 覆盖 abe【L3】 （答案④）
【(3, 0, 3)，(3, 1, 1)】 覆盖 acd【L4】

所以，测试用例的取法不唯一。注意有例外情形，例如，若把图例中第二个判断中的条件 x＞1 错写成 x＜1，那么利用上面两组测试用例，仍能得到同样结果。这表明，只是判定覆盖，还不能保证一定能查出在判断的条件中存在的错误。因此，还需要更强的逻辑覆盖准则去检验判断内部条件。

(3) 条件覆盖

所谓条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。例如在本题所给出的例子中，我们事先可对所有条件的取值加以标记。例如，

对于第一个判断：条件 A>1 取真值为$T1$，取假值为$\overline{T1}$
……条件 B=0 取真值为$T2$，取假值为$\overline{T2}$
对于第二个判断：条件 A=2 取真值为$T3$，取假值为$\overline{T3}$
……条件 x>1 取真值为$T4$，取假值为$\overline{T4}$

则可选取测试用例如下：

测试用例（答案⑤）	通过路径	条件取值	覆盖分支
【(2,0,4)，(2,0,3)】	ace(L1)	$T1$ $T2$ $T3$ $T4$	c,e
【(1,0,1)，(1,0,1)】	abd(L2)	$T1$ $T2$ $\overline{T3}$ $\overline{T4}$	b,d
【(2,1,1)，(2,1,2)】	abe(L3)	$T1$ $\overline{T2}$ $T3$ $\overline{T4}$	b,e

或

测试用例（答案②）	通过路径	条件取值	覆盖分支
【(1,0,3)，(1,0,4)】	abe(L3)	$T1$ $\overline{T2}$ $\overline{T3}$ $T4$	b,e
【(2,1,1)，(2,1,2)】	abe(L3)	$T1$ $\overline{T2}$ $T3$ $\overline{T4}$	b,e

注意，前一组测试用例不但覆盖了所有判断的取真分支和取假分支，而且覆盖了判断中所有条件的可能取值。但是后一组测试用例虽满足了条件覆盖，但只覆盖了第一个判断的取假分支和第二个判断的取真分支，不满足判定覆盖的要求。为解决这一矛盾，需要对条件和分支兼顾，有必要考虑以下的判定－条件覆盖。

(4) 判定－条件覆盖

所谓判定－条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断本身的所有可能判断结果至少执行一次。换言之，即是要求各个判断的所有可能的条件取值组合至少执行一次。例如，对于本题所给例子中的各判断，若$T1$、$T2$、$T3$、$T4$及$\overline{T1}$、$\overline{T2}$、$\overline{T3}$、$\overline{T4}$的含意如前所述，则只需设计以下两个测试用例便可覆盖图中的 8 个条件取值以及 4 个判断分支。

测试用例（答案①）	通过路径	条件取值	覆盖分支
【(2,0,4)，(2,0,3)】	ace(L1)	$T1$ $T2$ $T3$ $T4$	c,e
【(1,1,1)，(1,1,1)】	abd(L2)	$\overline{T1}$ $T2$ $T3$ $T4$	b,d

判定－条件覆盖也有缺陷。从表面上来看，它测试了所有条件的取值。但是事实并非如此。因为往往某些条件掩盖了另一些条件。对于条件表达式(A＞1）and（B＝0）来说，若（A＞1）的测试结果为真，则还要测试（B＝0），才能决定表达式的值；而若（A＞1）的测试结果为假，可以立刻确定表达式的结果为假。这时，往往就不再测试（B＝0）的取值了。因此，条件（B＝0）就没有检查。同样，对于条件表达式（A＝2）or（X＞1）来说，若（A＝2）的测试结果为真，就可以立即确定表达式的结果为真。这时，条件（X＞1）就没有检查。因此，采用判定－条件覆盖，逻辑表达式中的错误不一定能够查得出来。

(5) 条件组合覆盖

所谓条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。现在我们仍来考察本题所给出的例子，先对各个判断的条件取值组合加以标记。例如，

记 ① A＞1,B＝0 作$T1$ $T2$，属第一个判断的取真分支；
② A＞1,B≠0 作$T1$ $\overline{T2}$，属第一个判断的取假分支；
③ A≯1,B＝0 作$\overline{T1}$ $T2$，属第一个判断的取假分支；
④ A≯1,B≠0 作$\overline{T1}$ $\overline{T2}$，属第一个判断的取假分支；
⑤ A＝2,x＞1 作$T3$ $T4$，属第二个判断的取真分支；
⑥ A＝2,x≯1 作$T3$ $\overline{T4}$，属第二个判断的取真分支；
⑦ A≠2,x＞1 作$\overline{T3}$ $T4$，属第二个判断的取真分支；
⑧ A≠2,x≯1 作$\overline{T3}$ $\overline{T4}$，属第二个判断的取假分支；

对于每个判断，要求所有可能的条件取值的组合都必须取到。在图中的每个判断各有两个条件，所以各有 4 个条件取值的组合。我们取 4 个测试用例，就可用以覆盖上面 8 种条件取值的组合。必须明确，这里并未要求第一个判断的 4 个组合与第二个判断的 4 个组合再进行组合。要是那样的话，就需要 42＝16 个测试用例了。

测试用例（答案⑦）	通过路径	覆盖条件	覆盖组合号
【(2,0,4)，(2,0,3)】	ace(L1)	$T1$ $T2$ $T3$ $T4$	①,⑤
【(2,1,1)，(2,1,2)】	abe(L3)	$T1$ $\overline{T2}$ $T3$ $\overline{T4}$	②,⑥
【(1,0,3)，(1,0,4)】	abe(L3)	$\overline{T1}$ $T2$ $\overline{T3}$ $T4$	③,⑦
【(1,1,1)，(1,1,1)】	abd(L2)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ $\overline{T4}$	④,⑧

这组测试用例覆盖了所有条件的可能取值的组合，覆盖了所有判断的可取分支，但路径漏掉了 L4。测试还不完全。

(6) 路径测试

路径测试就是设计足够的测试用例，覆盖程序中所有可能的路径。若还是看本题所给出的例子，则可以选择如下的一组测试用例来覆盖该程序段的全部路径。

测试用例（答案⑥）	通过路径	覆盖条件
【(2,0,4)，(2,0,3)】	ace(L1)	$T1$ $T2$ $T3$ $T4$
【(1,1,1)，(1,1,1)】	abd(L2)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ $\overline{T4}$
【(1,1,2)，(1,1,3)】	abe(L3)	$\overline{T1}$ $T2$ $\overline{T3}$ $T4$
【(3,0,3)，(3,0,1)】	acd(L4)	$T1$ $T2$ $\overline{T3}$ $T4$

【例 6】下面是快速排序算法中的一趟划分算法，其中 datalist 是数据表，它有两个数据成员：一是元素类型为 Element 的数组 V，另一个是数组大小 n。算法中用到两个操作，一是取某数组元素 V[i]的关键码操作 getKey ( )，一是交换两数组元素内容的操作 Swap( )：

```
int Partition ( datalist &list, int low, int high ) {

//在区间[ low, high ]以第一个对象为基准进行一次划分，k 返回基准对象回放位置。

int k = low; Element pivot = list.V[low]; //基准对象

for ( int i = low+1; i <= high; i++ ) //检测整个序列，进行划分

if ( list.V[i].getKey ( ) < pivot.getKey( ) && ++ k != i )

Swap ( list.V[k], list.V[i] ); //小于基准的交换到左侧去

Swap ( list.V[low], list.V[k] ); //将基准对象就位

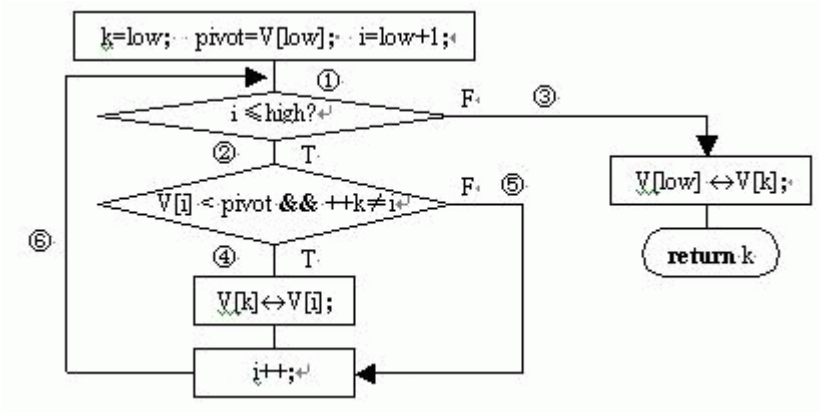
return k; //返回基准对象位置

}
```

(1) 试画出它的程序流程图；

(2) 试利用路径覆盖方法为它设计足够的测试用例（循环次数限定为 0 次，1 次和 2 次）。

答案：（1）流程图如下。



(2) 测试用例设计

循环次数	输入条件							输出结果					执行路径
	low	high	k	i	V[0]	V[1]	V[2]	k	i	V[0]	V[1]	V[2]	
0	0	0	0	1	-	-	-	0	1	-	-	-	①③
1	0	1	0	1	1	2	-	0	2	1	2	-	①②⑤⑥③
	0	1	0	1	2	1	-	1	2	1	2	-	①②④⑥③
	0	1	0	1	1	1	-	0	2	1	1	-	①②⑤⑥③
2	0	2	0	1	1	2	3	0	3	1	2	3	①②⑤⑥②⑤⑥③
	0	2	0	1	1	2	1	0	3	1	2	1	①②⑤⑥②⑤⑥③
	0	2	0	1	2	3	1	1	3	1	2	3	①②⑤⑥②④⑥③
	0	2	0	1	3	2	1	2	3	1	2	3	①②④⑥②④⑥③
	0	2	0	1	2	1	2	1	3	1	2	2	①②④⑥②⑤⑥③
	0	2	0	1	2	1	3	1	3	1	2	3	①②④⑥②⑤⑥③
	0	2	0	1	1	1	2	0	3	1	1	2	①②⑤⑥②⑤⑥③
	0	2	0	1	2	2	1	1	3	1	2	2	①②⑤⑥②④⑥③
	0	2	0	1	2	2	2	0	3	2	2	2	①②⑤⑥②⑤⑥③

分析：画程序流程图是设计测试用例的关键。从以往同学解题的经验来看，在画流程图时就出错了。所以首先要把流程图中的逻辑关系搞清楚再画出正确的流程图。考虑测试用例设计需要首先有测试输入数据，还要有预期的输出结果。对于此例，控制循环次数靠循环控制变量 i 和循环终值 high。循环 0 次时，取 low = high，此时一次循环也不做。循环一次时，取 low +1 = high，循环二次时，取 low+2 = high。

根据 BRO 策略，条件 V[i] < pivot && ++k≠i 的约束集合为 {(<, <), (<, =), (=, <), (>, <)}，因此，测试用例设计为：



四、习题

【5-1】从供选择的答案中选出应填入下列( )中的字句。

软件测试的目的是（ A ）。为了提高测试的效率，应该（ B ）。使用白盒测试方法时，确定测试数据应根据（ C ）和指定的覆盖标准。与设计测试数据无关的文档是（ D ）。

软件的集成测试工作最好由（ E ）承担，以提高集成测试的效果。

供选择的答案：

A. ① 评价软件的质量 ② 发现软件的错误

③ 找出软件中的所有错误 ④ 证明软件是正确的

B. ① 随机地选取测试数据

② 取一切可能的输入数据作为测试数据

③ 在完成编码以后制定软件的测试计划

④ 选择发现错误的可能性大的数据作为测试数据

C. ① 程序的内部逻辑 ② 程序的复杂程度

③ 使用说明书 ④ 程序的功能

D. ① 该软件的设计人员 ② 程序的复杂程度

③ 源程序 ④ 项目开发计划

E. ① 该软件的设计人员 ② 该软件开发组的负责人

③ 该软件的编程人员 ④ 不属于该软件开发组的软件设计人员

【5-2】请从供选择的答案中选出应填入下列（ ）中的字句。

程序的三种基本控制结构是（ A ）。它们的共同点是（ B ）。结构化程序设计的一种基本方法是（ C ）。软件测试的目的是（ D ）。软件调试的目的是（ E ）。

供选择的答案：

A. ① 过程，子程序，分程序 ② 顺序，条件，循环

③ 递归，堆栈，队列 ④ 调用，返回，转移

B. ① 不能嵌套使用 ② 只能用来写简单的程序 ③ 已经用硬件实现 ④ 只有一个入口和一个出口

C. ① 筛选法 ② 递归法 ③ 归纳法 ④ 逐步求精法

D. ① 证明程序中没有错误 ② 发现程序中的错误

③ 测量程序的动态特性 ④ 检查程序中的语法错误

E. ① 找出错误所在并改正之 ② 排除存在错误的可能性

③ 对错误性质进行分类 ④ 统计出错的次数

【5-3】从下列关于软件测试的叙述中，选出 5 条正确的叙述。

(1) 用黑盒法测试时，测试用例是根据程序内部逻辑设计的。

(2) 尽量用公共过程或子程序去代替重复的代码段。

(3) 测试是为了验证该软件已正确地实现了用户的要求。

(4) 对于连锁型分支结构，若有 n 个判定语句，则有 2n 条路径。

(5) 尽量采用复合的条件测试，以避免嵌套的分支结构。

(6) GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁。

(7) 发现错误多的程序模块，残留在模块中的错误也多。

(8) 黑盒测试方法中最有效的是因果图法。

(9) 在做程序的单元测试时，桩（存根）模块比驱动模块容易编写。

(10) 程序效率的提高主要应通过选择高效的算法来实现。

【5-4】从供选择的答案中选出同下列关于软件测试的各条叙述关系最密切的字句。

(1) 对可靠性要求很高的软件，例如操作系统，由第三者对源代码进行逐行检查。

(2) 已有的软件被改版时，由于受到变更的影响，改版前正常的功能可能发生异常，性能也可能下降。因此，对变更的软件进行测试是必要的。

(3) 在意识到被测试模块的内部结构或算法的情况下进行测试。

(4) 为了确认用户的需求，先做出系统的主要部分，提交给用户试用。

(5) 在测试具有层次结构的大型软件时，有一种方法是从上层模块开始，由上到下进行测试。此时，有必要用一些模块替代尚未测试过的下层模块。

供选择的答案：

A ~ E： ① 仿真器 ② 代码审查 ③ 模拟器 ④ 桩 ⑤ 驱动器

⑥ 域测试 ⑦ 黑盒测试 ⑧ 原型 ⑨ 白盒测试 ⑩ 退化测试

【5-5】对小的程序进行穷举测试是可能的，用穷举测试能否保证程序是百分之百正确呢？

【5-6】在任何情况下单元测试都是可能的吗？都是需要的吗？

【5-7】从供选择的答案中选出应填入下面有关软件测试的叙述的（ ）内的正确答案。

软件测试方法可分为黑盒测试法和白盒测试法两种。

黑盒测试法是通过分析程序的（ A ）来设计测试用例的方法。除了测试程序外，它还适用于对（ B ）阶段的软件文档进行测试。

白盒测试法是根据程序的（ C ）来设计测试用例的方法。除了测试程序外，它也适用于对（ D ）阶段的软件文档进行测试。

白盒法测试程序时常按照给定的覆盖条件选取测试用例。（ E ）覆盖比（ F ）覆盖严格，它使得每一个判定的每一条分支至少经历一次。（ G ）覆盖既是判定覆盖，又是条件覆盖，但它并不保证使各种条件都能取到所有可能的值。（ H ）覆盖比其他条件都要严格，但它不能保证覆盖程序中的每一条路径。

单元测试一般以（ I ）为主，测试的依据是（ J ）。

供选择的答案：

A, C：① 应用范围 ② 内部逻辑 ③ 功能 ④ 输入数据

B, D：① 编码 ② 软件详细设计 ③ 软件总体设计 ④ 需求分析

E, F, G, H：① 语句 ② 判定 ③ 条件 ④ 判定/条件

⑤ 多重条件 ⑥ 路径

I：① 白盒法 ② 黑盒法

J：① 模块功能规格说明 ② 系统模块结构图 ③ 系统需求规格说明

【5-8】从供选择的答案中选出应该填入下列关于软件测试的叙述的()内的正确答案。

软件测试中常用的静态分析方法是（ A ）和（ B ）。（ B ）用于检查模块或子程序间的调用是否正确。分析方法（白盒方法）中常用的方法是（ C ）方法。非分析方法（黑盒方法）中常用的方法是（ D ）方法和（ E ）方法。（ E ）方法根据输出对输入的依赖关系设计测试用例。

供选择的答案：

A ~ B： ① 引用分析 ② 算法分析 ③ 可靠性分析 ④ 效率分析

⑤ 接口分析 ⑥ 操作分析

C ~ E： ① 路径测试 ② 等价类 ③ 因果图 ④ 归纳测试

⑤ 综合测试 ⑥ 追踪 ⑦ 深度优先 ⑧ 调试

⑨ 相对图

【5-9】下面是选择排序的程序，其中 datalist 是数据表，它有两个数据成员：一是元素类型为 Element 的数组 V，另一个是数组大小 n。算法中用到两个操作，一是取某数组元素 V[i]的关键码操作 getKey ( )，一是交换两数组元素内容的操作 Swap( )：：

```
void SelectSort ( datalist & list ) {

//对表 list.V[0]到 list.V[n-1]进行排序, n 是表当前长度。

for ( int i = 0; i < list.n-1; i++ ) {

int k = i; //在 list.V[i].key 到 list.V[n-1].key 中找具有最小关键码的对象

for ( int j = i+1; j < list.n; j++)

if ( list.V[j].getKey ( ) < list.V[k].getKey ( ) ) k = j; //当前具最小关键码的对象

if ( k != i ) Swap ( list.V[i], list.V[k] ); //交换

}

}
```

(1) 试计算此程序段的 McCabe 复杂性；

(2) 用基本路径覆盖法给出测试路径；

(3) 为各测试路径设计测试用例。

【5-10】根据下面给出的规格说明，利用等价类划分的方法，给出足够的测试用例。

“一个程序读入三个整数。把此三个数值看成是一个三角形的三个边。这个程序要打印出信息，说明这个三角形是三边不等的、是等腰的、还是等边的。”

【5-11】设要对一个自动饮料售货机软件进行黑盒测试。该软件的规格说明如下：

“有一个处理单价为 1 元 5 角钱的盒装饮料的自动售货机软件。若投入 1 元 5 角硬币，按下“可乐”、“雪碧”或“红茶”按钮，相应的饮料就送出来。若投入的是 2 元硬币，在送出饮料的同时退还 5 角硬币。”

(1) 试利用因果图法，建立该软件的因果图；

(2) 设计测试该软件的全部测试用例。

【5-12】对一个长度为 100,000 条指令的程序进行测试，记录下来的数据如下：

ú 测试开始，发现错误个数为 0；

ú 经过 160 小时的测试，累计改正 100 个错误，此时，MTTF = 0.4 小时；

ú 又经过 160 小时的测试，累计改正 300 个错误，此时，MTTF = 2 小时；

(1) 估计程序中固有的错误总数；

(2) 为使 MTTF 达到 10 小时，必须测试和调试这个程序多长时间?

(3) 给出 MTTF 与测试时间 t 之间的函数关系。

【5-13】应该由谁来进行确认测试？是软件开发者还是软件用户？为什么？

五、习题解答

【5-1】A. ② B. ④ C. ① D. ④ E. ④

软件测试的目的是软件中的错误。因为不可能把所有可能的输入数据都拿来测试（时间花费不起），为了提高测试的效率，应该选择发现错误的可能性大的数据作为测试数据。

使用白盒测试方法时，确定测试数据应根据程序的内部逻辑和指定的覆盖标准，可以不考虑程序的功能。与设计测试数据无关的文档是项目开发计划。

软件的集成测试工作最好由不属于该软件开发组的软件设计人员承担，以提高集成测试的效果。

【5-2】A. ② B. ④ C. ④ D. ② E. ①

1966 年，Bohm 与 Jacopini 提出任何单入口单出口的没有“死循环”的程序都能由三种最基本的控制结构构造出来。这三种基本控制结构就是“顺序结构”、“选择 IF－THEN－ELSE 结构”、“重复 DO－WHILE 或 DO－UNTIL 结构”。 它们的共同点是只有一个入口和一个出口。E.W.Dijkstra 提出了程序要实现结构化的主张，并将这一类程序设计称为结构化程序设计。这种方法的一个重要原则就是采用自顶向下、逐步求精的方法编写程序。N.Wirth 曾做过如下说明：“我们对付一个复杂问题的最重要的方法就是抽象。 因此，对于一个复杂的问题，不要急于马上用计算机指令、数字和逻辑符号来表示它，而应当先用较自然的抽象的语句来表示，从而得到抽象的程序。抽象程序对抽象的数据类型进行某些特定的运算，并用一些合适的记号（可以是自然语言）来表示。下一步对抽象程序再做分解，进入下一个抽象的层次。这样的细化过程一直进行下去，直到程序能被计算机接受为止。此时的程序已经是用某种高级语言或机器指令书写的了。”

软件调试则是在进行了成功的测试之后才开始的工作。它与软件测试不同，软件测试的目的是尽可能多地发现软件中的错误，但进一步诊断和改正程序中潜在的错误，则是调试的任务。调试活动由两部分组成：① 确定程序中可疑错误的确切性质和位置。② 对程序（设计，编码）进行修改，排除这个错误。

【5-3】正确的叙述有(4)、(5)、(6)、(7)、（10）。

黑盒测试主要是根据程序的有关功能规格说明和覆盖准则来设计测试用例，进行测试的，不是根据程序的内部逻辑来设计测试用例，这是白盒测试做的事情。在所有黑盒测试方法中，最有效的不是因果图法，而是边界值分析方法。测试的目的是尽可能多地发现软件中的错误，其附带的收获才是验证该软件已正确地实现了用户的要求。测试的一条重要原则是：发现错误多的程序模块，残留在模块中的错误也多。软件可靠性模型（Shooman）就是依据这个原则建立它的公式的。对于连锁型分支结构，若有 n 个判定语句，则有 2n 条路径。因此，随着 n 的增大，路径数增长非常快。单元测试时，因为桩模块要模拟子模块的功能，这不是一件容易的事情，而驱动模块只是控制被测模块的执行，所以桩模块的编写比驱动模块的编写要难得多。

在程序设计风格方面，如果重复的代码段没有明显的功能，不可以抽取出来形成独立的公共过程或子程序，只有在这些代码段表现出独立的功能时，才可把它们抽取出来形成独立的公共过程或子程序。另外，程序效率的提高主要应通过选择高效的算法或使用高效的语言编译器来实现。GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁，这句话是正确的。

【5-4】(1) ② (2) ⑩ (3) ⑨ (4) ⑧ (5) ④

(1) 对可靠性要求很高的软件，由第三者对源代码进行逐行检查，这是代码审查。

(2) 软件变更时可能发生退化现象：原来正常的功能可能发生异常，性能也可能下降。因此，对变更的软件要进行退化测试。

(3) 基于被测试模块的内部结构或算法设计测试用例进行测试，这是白盒测试。
(4) 为了确认用户的需求，先做出系统的原型，提交给用户试用。

(5) 自顶向下对具有层次结构的大型软件进行集成测试时，需要设计一些虚拟模块来替代尚未测试过的下层模块，这些模块叫做桩模块。

【5-5】对小程序进行穷举测试，不见得能保证程序百分之百正确。所谓穷举测试是拿所有可能的输入数据来作为测试用例（黑盒测试），或覆盖程序中所有可能的路径（白盒测试）。对于小程序来说，实际上并不能真正作到穷举测试。例如前面讲过，一个小程序 P 只有两个输入 X 和 Y 及输出 Z，在字长为 32 位的计算机上运行。如果 X、Y 只取整数，考虑把所有的 X、Y 值都做为测试数据，按黑盒方法进行穷举测试，这样做可能采用的测试数据组（Xi，Yi），基数（radix）i 的最大可能数目为：232×232＝264。如果程序 P 测试一组 X、Y 数据需要 1 毫秒，而且假定一天工作 24 小时，一年工作 365 天，要完成 264 组测试，需要 5

亿年。
【5-6】单元测试又称模块测试，是针对软件设计的最小单位—程序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

单元测试是在编码阶段完成的，每编写出一个程序模块，就开始做这个模块的单元测试，所以只要采用模块化方法开发软件，单元测试都是必需的。它可由编写程序的人来完成。因为它需要根据程序的内部结构设计测试用例，对于那些不了解程序内部细节的人，这种测试无法进行。

【5-7】A. ③ B. ④ C. ② D. ② E. ② F. ① G. ④ H. ⑤ I. ① J. ①

软件测试方法可分为黑盒测试法和白盒测试法两种。黑盒测试法是基于程序的功能来设计测试用例的方法。除了测试程序外，它还适用于对需求分析阶段的软件文档进行测试。白盒测试法是根据程序的内部逻辑来设计测试用例的方法。除了测试程序外，它也适用于对软件详细设计阶段的软件文档进行测试。

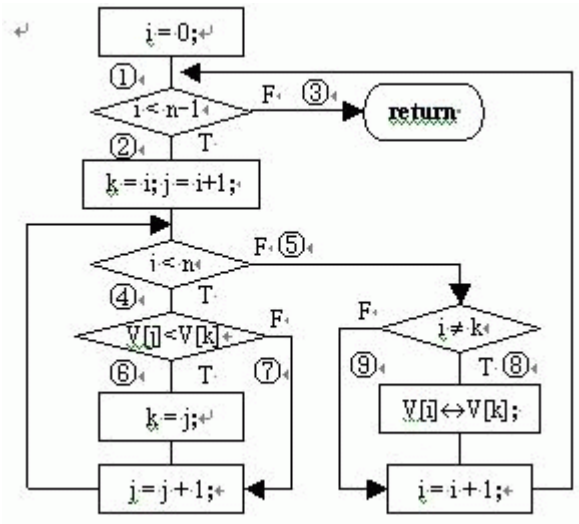
白盒法测试程序时常按照给定的覆盖条件选取测试用例。判定覆盖比语句覆盖严格，它使得每一个判定的每一条分支至少经历一次。判定/条件覆盖既是判定覆盖，又是条件覆盖，但它并不保证使各种条件都能取到所有可能的值。多重条件覆盖，也叫组合条件覆盖，比其他条件都要严格，但它不能保证覆盖程序中的每一条路径。

单元测试一般以白盒法为主，测试的依据是系统的模块功能规格说明。

【5-8】A. ① B. ⑤ C. ① D. ② E. ③

软件测试中常用的静态分析方法是引用分析和接口分析。接口分析用于检查模块或子程序间的调用是否正确。分析方法（白盒方法）中常用的方法是路径测试方法。非分析方法（黑盒方法）中常用的方法是等价类（划分）方法和因果图方法。因果图方法根据输出对输入的依赖关系设计测试用例。

【5-9】



(1) McCabe 环路复杂性 = 5

(2) 独立路径有 5 条：

①③

①②⑤⑧……

①②⑤⑨……

①②④⑥……

①②④⑦……

(3) 为各测试路径设计测试用例:

路径①③：取 n = 1

路径①②⑤⑧……：取 n = 2,

预期结果：路径⑤⑧③不可达

路径①②⑤⑨……：取 n = 2,

预期结果：路径⑤⑨③不可达

路径①②④⑥⑤⑧③：

取 n = 2, V[0] = 2, V[1] = 1, 预期结果：k = 1, V[0] = 1, V[1] = 2

路径①②④⑥⑤⑨③：

取 n = 2, V[0] = 2, V[1] = 1, 预期结果：k = 1, 路径⑨③不可达

路径①②④⑦⑤⑧③：

取 n = 2, V[0] = 1, V[1] = 2, 预期结果：k = 0, 路径⑧③不可达

路径①②④⑦⑤⑨③：

取 n = 2, V[0] = 1, V[1] = 2, 预期结果：k = 0, V[0] = 1, V[1] = 2

【5-10】设三角形的三条边分别为 A, B, C。如果它们能够构成三角形的三条边，必需满足： A > 0，B > 0，C > 0，且 A + B > C，B + C > A，A + C > B。

如果是等腰的，还要判断是否 A = B，或 B = C，或 A = C。

对于等边的，则需判断是否 A = B，且 B = C，且 A = C。

列出等价类表：

输入条件	有效等价类	无效等价类
是否三角形的三条边	(A > 0) (1), (B > 0) (2), (C > 0) (3), (A + B > C), (4) (B + C > A) (5), (A + C > B) (6)	A ≤ 0 (7), B ≤ 0 (8), C ≤ 0 (9), A + B ≤ C (10), A + C ≤ B (11), B + C ≤ A (12)
是否等腰三角形	(A = B) (13), (B = C) (14), (A = C) (15)	(A ≠ B) and (B ≠ C) and (A ≠ C) (16)
是否等边三角形	(A = B) and (B = C) and (A = C) (17)	(A ≠ B) (18), (B ≠ C) (19), (A ≠ C) (20)

设计测试用例：输入顺序是〖A，B，C〗

- 〖3，4，5〗覆盖等价类 (1),(2),(3),(4),(5),(6)。满足即为一般三角形。
- 〖0，1，2〗覆盖等价类 (7)。不能构成三角形。
- 〖1，0，2〗覆盖等价类 (8)。同上。
- 〖1，2，0〗覆盖等价类 (9)。同上。

（若不考虑特定 A,B,C，三者取一即可）

- 〖1，2，3〗覆盖等价类 (10)。同上。
- 〖1，3，2〗覆盖等价类 (11)。同上。
- 〖3，1，2〗覆盖等价类 (12)。同上。

（若不考虑特定 A,B,C，三者取一即可）

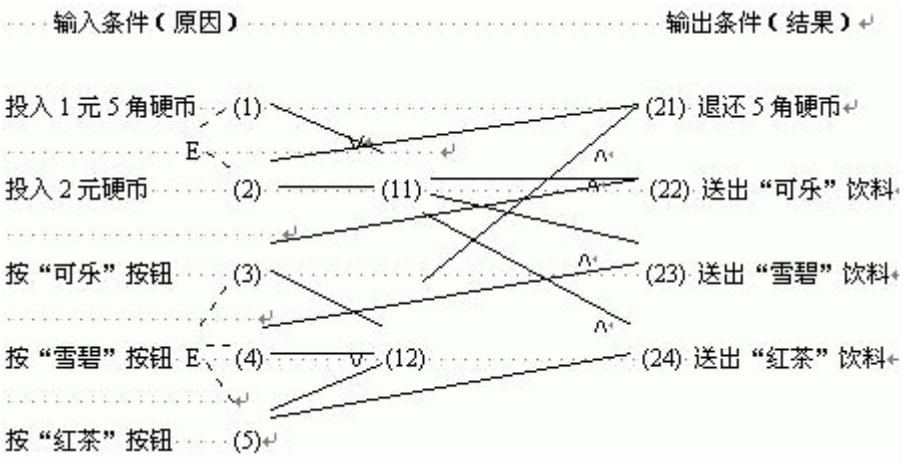
- 〖3，3，4〗覆盖等价类 (1),(2),(3),(4),(5),(6),(13)。
- 〖3，4，4〗覆盖等价类 (1),(2),(3),(4),(5),(6),(14)。
- 〖3，4，3〗覆盖等价类 (1),(2),(3),(4),(5),(6),(15)。

（满足即为等腰三角形，若不考虑特定 A,B,C，三者取一即可）

- 〖3，4，5〗覆盖等价类 (1),(2),(3),(4),(5),(6),(16)。不是等腰三角形。
- 〖3，3，3〗覆盖等价类 (1),(2),(3),(4),(5),(6),(17)。是等边三角形
- 〖3，4，4〗覆盖等价类 (1),(2),(3),(4),(5),(6),(14),(18)。
- 〖3，4，3〗覆盖等价类 (1),(2),(3),(4),(5),(6),(15),(19)。
- 〖3，3，4〗覆盖等价类 (1),(2),(3),(4),(5),(6),(13),(20)。

（不是等边三角形，若不考虑特定 A,B,C，三者取一即可）

【5-11】(1) 因果图



(2) 测试用例设计

			1	2	3	4	5	6	7	8	9	10	11
输入	投入 1 元 5 角硬币	(1)	1	1	1	1	0	0	0	0	0	0	0
	投入 2 元硬币	(2)	0	0	0	0	1	1	1	1	0	0	0
	按“可乐”按钮	(3)	1	0	0	0	1	0	0	0	1	0	0
	按“雪碧”按钮	(4)	0	1	0	0	0	1	0	0	0	1	0
	按“红茶”按钮	(5)	0	0	1	0	0	0	1	0	0	0	1
中间结点	……已投币	(11)	1	1	1	1	1	1	1	1	0	0	0
	……已按钮	(12)	1	1	1	0	1	1	1	0	1	1	1
输出	退还 5 角硬币	(21)	0	0	0	0	1	1	1	0	0	0	0
	送出“可乐”饮料	(22)	1	0	0	0	1	0	0	0	0	0	0
	送出“雪碧”饮料	(23)	0	1	0	0	0	1	0	0	0	0	0
	送出“红茶”饮料	(24)	0	0	1	0	0	0	1	0	0	0	0

测试用例 每一纵列为一个测试用例

【5-12】由 shooman 公式：

$$MTTF = \frac{I_T}{K(E_T - E_C(T))}$$

其中，$I_T=10^5$，$MTTF_1=0.4$，$T_1=160$，$n_1=100$，$MTTF_2=2$，$T_2=320$，$n_2=300$。得：

$$\begin{cases} MTTF_1 = \dfrac{I_T}{K(E_T - E_C(T_1))} \\ MTTF2 = \dfrac{I_T}{K(E_T - E_C(T_2))} \end{cases} \qquad \begin{cases} 0.4 = \dfrac{100000}{K(E_T - 100)} \\ 2 = \dfrac{100000}{K(E_T - 300)} \end{cases}$$

(1) → 解得程序中固有得错误总数 $E_T=350$，此外 $K=1000$。

(2) → 设 MTTF=10，有

$$10 = \frac{100000}{1000(350 - x)}$$

解得 $x=340$。由可靠性累积曲线 $E_C(t)=E_T(1-e^{-K1t})$，

得 $\begin{cases} 100=350(1-e^{-160K1}) \\ 300=350(1-e^{-320K1}) \end{cases}$

解得 $K1\approx0.01$。代入：$340=350(1-e^{-K1t})=350(1-e^{-0.01t})$

$t=(\ln(35))/0.01\approx356$(小时)

因此求得为使 MTTF=10，测试和调试该程序需要花费 356 小时。

(3) MTTF 与测试时间 t 之间的函数关系：

因为 $E_C(t)=E_T(1-e^{-K1t})$，则 $E_T-E_C(t)=E_Te^{-K1t}$。代入 shooman 公式：

$$MTTF = \frac{I_T}{K(E_T - E_r(t))} = \frac{I_T}{K \cdot E_T e^{-K1t}} = \frac{100000}{1000 \cdot 350 e^{-0.01t}} = \frac{2}{7e^{-0.01t}}$$

【5-13】在对照需求做有效性测试和软件配置审查时，是由软件开发者在开发环境下进行的测试。而接下来做验收测试时则以用户为主。软件开发人员和 QA（质量保证）人员也应参加。由用户参加设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果。一般使用生产中的实际数据进行测试。

如果软件是为多个客户开发的，则需要进行 α 测试和 β 测试。α 测试是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。软件在一个自然设置状态下使用。开发者坐在用户旁边，随时记下错误情况和使用中的问题。这是在受控制的环境下进行的测试。 β 测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。这些用户是与公司签定了支持产品预发行合同的外部客户，他们要求使用该产品，并愿意返回有关错位错误信息给开发者。与 α 测试不同的是，开发者通常不在测试现场。因而， β 测试是在开发者无法控制的环境下进行的软件现场应用。

第六章 面向对象的软件工程

一、复习要求

1. 了解面向对象的概念，包括什么是范型，面向对象的概念，对象和对象的分类等。

2. 了解用面向对象方法构造软件的开发过程，包括应用生存期和类生存期的概念。

3. 了解面向对象分析方法，包括论域分析，应用分析的介绍。

4. 了解面向对象设计方法，包括高层设计模型和设计原则，类设计的目标和方针，复用设计及类设计的方法。

5. 了解有影响的 coad 方法、Booch 方法和 OMT 方法的基本思想。

二、内容提要

面向对象技术是一个非常实用而强有力的软件开发方法。它的特征是：

§ 方法的唯一性，即方法是对软件开发过程所有阶段进行综合考虑而得到的。

§ 从生存期的一个阶段到下一个阶段的高度连续性，即生存期后一阶段的成果只是在前一阶段成果的补充和修改。

§ 把面向对象分析(OOA)、面向对象设计(OOD)和面向对象程序设计(OOP)集成到生存期的相应阶段。

1. 面向对象的概念

(1) 范型

范型(Paradigm)又称为范例、风范或模式(Pattern)。从软件开发角度来看，范型与问题解决技术有关。范型定义了特定的问题和应用的开发过程中将要遵循的步骤，确定将用于表示问题和它的解决的那些成分的类型，并利用这些成分表示与问题解决有关的抽象，直接得到问题的结构。因此，范型的选择影响整个软件开发生存期。就是说，它支配了设计方法、编码语言、测试和检验技术的选择。

① 流行的范型 ：目前流行多种范型，它们提供了许多方法，可进行系统分解。流行的范型有：过程性的，逻辑的，面向存取的，面向进程的，面向对象的，函数型的，说明性的。每个范型都有它的支持者和用户，每个范型都特别适合于某种类型的问题或子问题。例如，逻辑程序设计范型是基于规则的，它把有关问题的知识分解成一组具体规则，用语言的“if_then”等结构来表示这些规则。面向存取范型是一种在构造用户界面方面很有用的技术。此外，每一个范型都用不同的方式考虑问题，每一个范型都使用不同的方法来分解问题，而且每一个范型都导致不同种类的块、过程、产生规则。下面主要讨论三种范型。研究的目的是帮助我们找到解决问题的入手点。

② 过程性范型 ：过程性范型是使用最广泛、历史最长的软件范型。它产生过程的抽象，这些抽象把软件视为处理流，并定义成由一系列步骤构成的算法。每一步骤都是带有预定输入和特定输出的一个过程，把这些步骤串联在一起可产生合理的稳定的贯通于整个程序的控制流，最终产生一个简单的具有静态结构的体系结构，如图 6.1(a)所示。

过程性范型侧重建立构成问题解决的处理流，数据抽象、数据结构是根据算法步骤的要求开发的，它贯穿于过程，提供过程所要求操作的信息。系统的状态是一组全局变量，这组全局变量保存状态的值，把它们从一个过程传送到另一个过程。

过程性范型是一种成熟的应用开发过程。对这种方法已有许多支持工具。然而，在大型系统的开发上存在一些问题。

③ 面向对象范型 ：在过程性范型中优先考虑的是过程抽象，而在面向对象范型中优先考虑的是实体，即问题论域的对象。在面向对象范型中，把标识和模型化问题论域中的主要实体做为系统开发的起点，主要考虑对象的行为而不是必须执行的一系列动作。 面向对象系统中的对象是数据抽象与过程抽象的综合。系统的状态保存在各个数据抽象的核心所定义的数据存储中。控制流包含在各个数据抽象中的操作内。不像在过程性范型里那样，把数据从一个过程传送到另一个过程，而是把控制流从一个数据抽象通过消息传送到另一个数据抽象。完成的系统体系结构更复杂但也更灵活，如图 6.2(b)所示。把控制流分离成块，这样可以把复杂的动作视为各个局部间的相互作用。

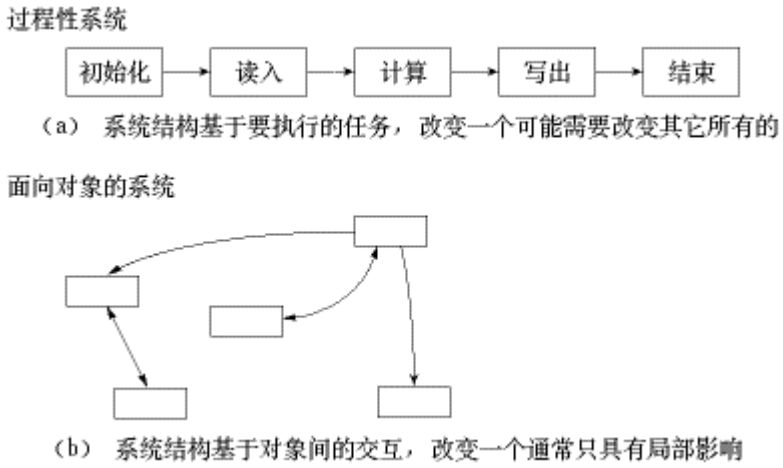


图 6.1 过程性系统和面向对象系统的基本构造

④ 面向进程的范型 ：面向进程的范型是把一个问题分解成独立执行的模块。让不只一个程序同时运行。这些进程互相配合，解决问题。面向进程范型产生的主要的块是进程。一个进程中的活动独立于其它进程的活动，但可以要求从其它进程得到信息，或为其它进程提供信息。甚至可以异步处理，仅需要进程暂停发送或接收信息。在面向对象范型中，各个对象是相对独立的，但也存在单线索(单线程)控制。面向进程范型支持与面向对象范型相同的封装，但可提供多线索(多线程)执行。

⑤ 混合范型 ：在大型系统的开发中，很难说哪种范型对整个问题的解决最好。系统开发现在有一种补充步骤，可把大型问题分解成一组子问题。对于每个子问题可以采用适当的软件范型。例如，设计一个智能数据分析系统时，可把它分解为 4 个子系统，如图 6.2 所示。系统的数据库界面，可以使用面向对象的方法进行设计；智能数据分析用逻辑范型设计；而分析算法则是过程性的；系统通过一个用户界面来实用化，这个用户界面是用面向存取范型设计出来的。

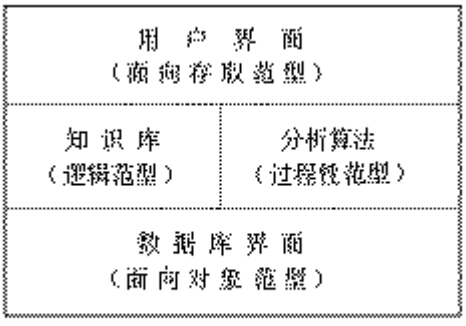


图 6.2 一个智能数据分析系统

这种设计需要有某种实现语言或一组协同语言的支持。许多流行的功能不断增强的语言支持不只一种设计范型。对于混合范型，现在已经存在不少技术。像 C++

和并发 C 这样的语言都是多范型语言，支持过程性范型和面向对象范型。并发 C 还支持面向进程范型。系统可以使用单一的语言，利用两种或多种范型写成。还可以利用可共享数据格式和连接规约的某些语言，把用这些语言分别编写的块链接到某个单一的应用中去。

(2) 面向对象的概念

关于“面向对象”，有许多不同的看法。Coad 和 Yourdon 给出了一个定义：“面向对象 = 对象 + 类 + 继承 + 消息通信”。如果一个软件系统是使用这样 4 个概念设计和实现的，则认为这个软件系统是面向对象的。一个面向对象的程序的每一成份应是对象，计算是通过新的对象的建立和对象之间的消息通信来执行的。

(3) 对象(object)

一般意义来讲，对象是现实世界中存在的一个事物。可以是物理的，如一个家具或桌子，如图 6.3 所示，可以是概念上的，如一个开发项目。对象是构成现实世界的一个独立的单位，具有自己的静态特征(用数据描述)和动态特征(行为或具有的功能)。 例如：人的特征：姓名、性别、年龄等，行为：衣、食、住、行等。

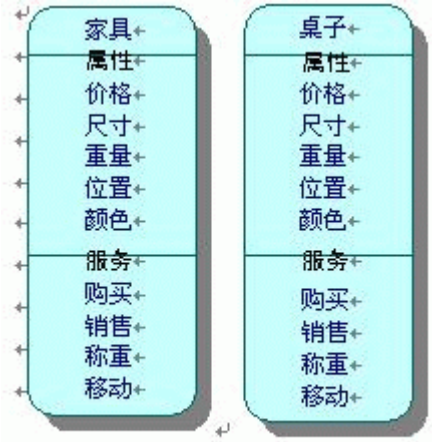


图 6.3 对象的定义

① 对象、属性、操作、消息定义

对象可以定义为系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位，由一组属性和一组对属性进行操作的服务组成。

属性一般只能通过执行对象的操作来改变。

操作又称为方法或服务，在 C++中称为成员函数，它描述了对象执行的功能，若通过消息传递，还可以为其它对象使用。

而所谓的消息是一个对象与另一个对象的通信单元，是要求某个对象执行类中定义的某个操作的规格说明。发送给一个对象的消息定义了一个操作名和一个参数表(可能是空的)，并指定某一个对象。由一个对象接收的消息则调用消息中指定的操作，并将传递过来的实际参数与参数表中相应的形式参数结合起来。接收对象对消息的处理可能会改变对象中的状态，即改变接收对象的属性，并发送一个消息给自己或另一个对象。可以认为，这种消息的传递大致等价于过程性范型中的函数调用。

② 对象的分类

§ 外部实体：与软件系统交换信息的外部设备、相关子系统、操作员或用户等。

§ 信息结构：问题信息域中的概念实体，如信号、报表、显示信息等。

§ 需要记忆的事件：在系统运行过程中可能产生并需要系统记忆的事件，如单击鼠标左键、击打键盘“?”键等。

§ 角色：与软件系统交互的人员所扮演的角色，如经理、部长、技术支持等。

§ 组织机构：有关机构，如单位、小组等。

§ 位置：作为系统环境或问题上下文的场所、位置，如客户地址、收件人(机构)地址等。

§ 操作规程：如操作菜单、某种数据输入过程等。

在标识对象时必需注意遵循“信息隐蔽”的原则：必需将对象的属性隐藏在对象的内部，使得从对象的外部看不到对象的信息是如何定义的，只能通过该对象界面上的操作来使用这些信息。对象的状态通过给对象赋予具体的属性值而得到。它只能通过该对象的操作来改变。 对象有两个视图，分别表现在分析设计和实现方面。从分析及设计方面来看，对象表示了一种概念，它们把有关的现实世界的实体模型化。从实现方面来看，一个对象表示了在应用程序中出现的实体的实际数据结构。之所以有两个视图，是为了把说明与实现分离，对数据结构和相关操作的实现进行封装。

(4) 类(class)和实例(instance)

把具有相同特征和行为的对象归在一起就形成了类。类成为某些对象的模板，抽象地描述了属于该类的全部对象的属性和操作。属于某个类的对象叫做该类的实例。对象的状态则包含在它的实例变量，即实例的属性中。如图 6.4 所示。从“李杰”、“王辉”和“杨芳”等对象可得到类“学生”，而这些对象就称为该类的实例。

类定义了各个实例所共有的结构，类的每一个实例都可以使用类中定义的操作。实例的当前状态是由实例所执行的操作定义的。

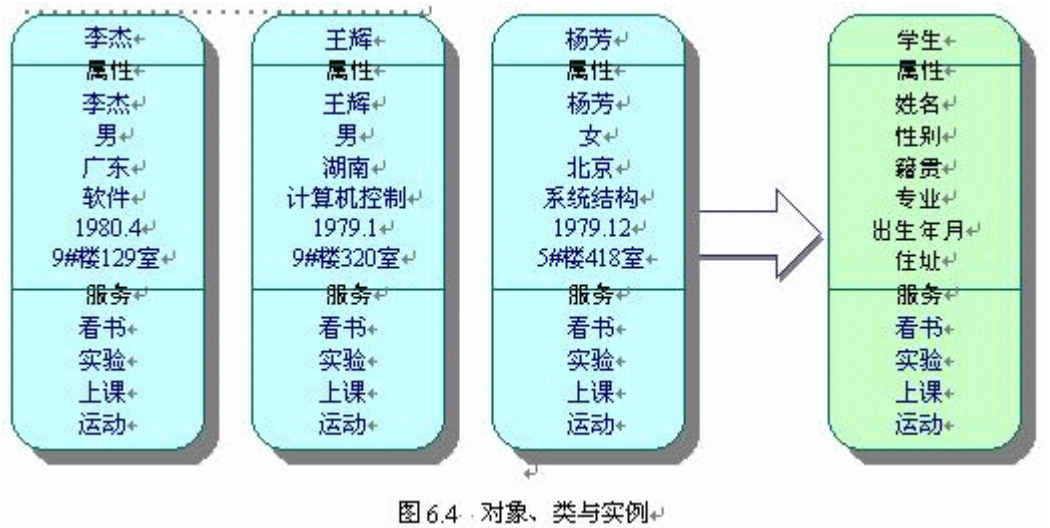


图 6.4 对象、类与实例

面向对象程序设计语言，如 C++和 smalltalk 都定义了一个 new 操作，可建立一个类的新实例。C++还引入了构造函数，用它在声明一个对象时建立实例。此外，

程序设计语言给出了不同的方法，来撤消(称为析构)实例，即当某些对象不再使用时把它们删去，把存储释放以备其它对象使用。C++给出了一个操作 delete，可以释放一个对象所用的空间。C++还允许每个类定义自己的析构方法，在撤消一个对象时调用它。smalltalk 没有提供一个机制来撤消对象，但可以进行无用单元收集。

类常常可看做是一个抽象数据类型(ADT)的实现。但更重要的是把类看做是表示某种概念的一个模型。事实上，类是单个的语义单元，它可以很自然地管理系统中的对象，匹配数据定义与操作。类加进了操作，给通常的记录赋予了语义，可提供各种级别的可访问性。

(5) 继承 (inheritance)

如果某几个类之间具有共性的东西(信息结构和行为)，抽取出来放在一个一般类中，而将各个类的特有的东西放在特殊类中分别描述，则可建立起特殊类对一般类的继承。如图 6.5 所示，。各个特殊类可以从一般类中继承共性，这样避免了重复。

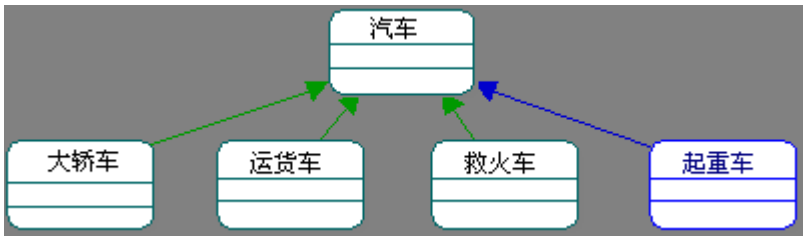


图 6.5 特殊类对一般类的继承关系

建立继承结构的好处：

§ 易编程、易理解 代码短， 结构清晰；
§ 易修改：共同部分只要在一处修改即可；
§ 易增加新类：只须描述不同部分。

(6) 多继承

如果一个类需要用到多个既存类的特征，可以从多个类中继承，称为多继承。例如退休教师是继承退休者和教师这两个类的某些特征或行为而得到的一个新类。

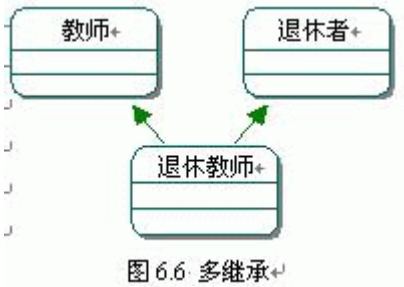


图 6.6 多继承

(7) 多态性和动态绑定

对象互相通信，即一个对象发消息给另一个对象，执行某些行为或又发消息给另外的对象，从而执行系统的功能。
发送消息的对象可能不知道另一个对象的类型是什么。如在 C 程序中使用命令 ClearInt（）时要严格区分该命令适合一个整数，还是一个整数数组。但在 C++情形，ClearInt（）对两者都适用，它自己判断对象是哪一个。
这就是多态性。它意味着一个操作在不同类中可以有不同的实现方式。如清零操作 ClearInt（）针对消息对象是 int array 还是 int，其实现是不同的。在一个面向对象的多态性语言中，可能代替一个特定类型的类型的集合就是它的子类集合。

例如，图 6.7 给出了 4 个类的继承层次。使用这个继承结构，发送给多边形类的所有消息，它的所有子类都能够响应。又例如，想要在屏幕上画一系列多边形，多态性允许一个表的元素可以属于一组指定的类型而不仅仅是一个类型，可以认为这是一个类族。通过遍历这个表，发送给各个表元素以 draw 消息，画出所有的多边形。

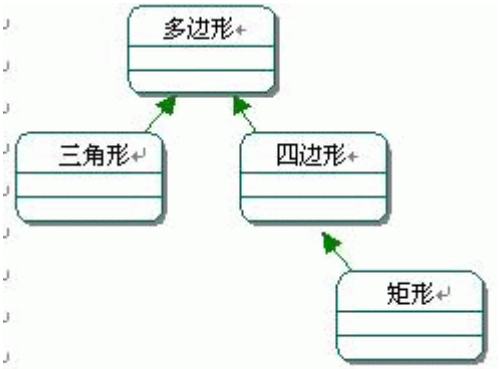


图 6.7 4 个类的继承层次

动态绑定把函数调用与目标代码块的连接延迟到运行时进行。这样，只有发送消息时才与接收消息实例的一个操作绑定。它与多态性可以使我们建立的系统更灵活，易于扩充。做为动态绑定的例子，考虑在多边形类中的方法 contains?(aPoint)。这个操作可以在类层次的各层重新实现，以有效利用各个子类的特殊的特征。例如，假定一个矩形有某些边与屏幕的边平行，这时，检查一个点是否包含在矩形内，比检查一个点是否在一个一般的四边形内的效率要高一些。
2. 面向对象软件的开发过程

面向对象范型不仅是一些具体的软件开发技术与策略，而且是一整套关于如何看待软件系统与现实世界的关系以及如何进行系统构造的软件方法学。用面向对象开发方法构造的软件具有以下特点：
§ 面向对象的技术建立的模型与客观世界一致，因而便于理解；

§ 适应变化的需要，修改局限在模块中；
§ 可复用性。
(1) 应用生存期

图 6.8 给出应用生存期模型。在图中各个阶段的顺序是线性的，但实际上开发过程不是线性的。还没有办法用图来逼真地反映在面向对象开发过程中各个阶段之

间的复杂交互。有一部分分析工作在设计之前实行，但有些分析工作与其它部分的设计与实现并行进行。

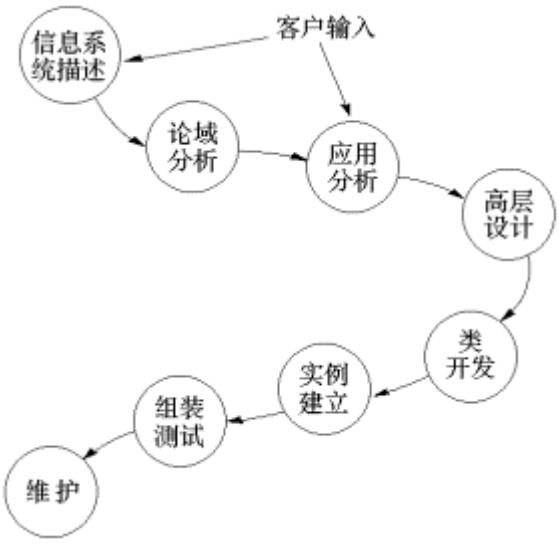


图 6.8 一个基于复用的应用生存期

开发可复用的软件构件是软件开发过程的一部分。面向对象方法以类作为单元，并分别考虑类的生存期与应用生存期。类生存期可包含在图 6.8 中的类开发阶段中，可与应用生存期集成。

(2) 类生存期

在面向对象软件开发过程中特别重视复用。软件构件应独立于当初开发它们的应用而存在。构件的开发瞄准某些局部的设计和实现，它们可用于当前问题的解决，但为了在以后的项目中使用，它们还应当足够通用。在以后的应用开发中，可以调整这些独立构件以适应新问题的需要。因此，应使得类成为一个可复用的单元，图 6.9 提出了一个类生存期。

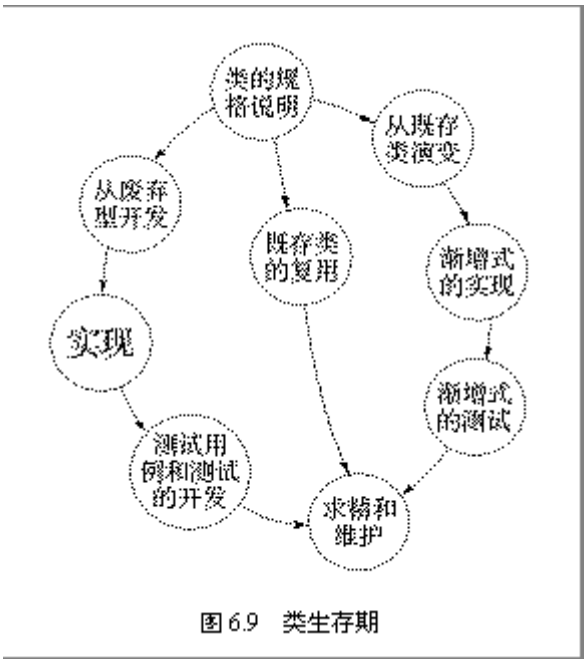


图 6.9 类生存期

类生存期与应用生存期交叉。在应用生存期的每一个阶段都可做类的标识。类生存期有自己的步骤，与任一特定应用的开发无关。按照这些步骤，可以完整地描述一个基本实体。而不仅仅考虑当前正在开发的系统。系统开发的各个阶段都可能会标识新的类。随着各个新类的标识，类生存期引导开发工作逐个阶段循序渐进。

例如，在应用分析中已经标识了对一个图形显示设备的要求。如果这样一个图形显示设备类不存在，就应着手开发。但是，用到显示器所有可能操作的应用寥寥无几。若把这些操作的开发当做一个特定应用系统开发的一部分，那么只可能标识和实现该系统所要求的那些操作。但如果考虑让构件独立于应用，就必须能够综合出超出当前系统需求的开发要求，生成一种能表示成一个完全的概念的模型并可建立为以后其它系统复用的类。

在纯面向对象的系统开发中，一个应用程序就“是”一个类。基本的类，像 list 类，可不涉及应用，但基本类的实例要聚合到其它类的定义中。这些类依次又聚合到更复杂的类定义中，最终将会遇到一个类，它涉及整个应用。

下面概括了类生存期各个阶段主要做的事情。

① 类的规格说明 ：对每一个类都要开发它的规格说明，无论是在哪一个阶段标识的类都是如此。类的规格说明定义了施加于对象的数据存储上的一组操作。这组操作应工作在封装在对象内部的数据存储上，或返回关于对象状态的信息。操作的名字应能反映这个操作本身的含义。类的规格说明必须足够完整，使得它能够与在类资源库中的那些可复用的类的规格说明做比较。

② 类的设计与实现 ：此时尽可能利用既存类提供为当前应用所需要的功能。图 6.9 给出了利用既存类的三个途径：

§ 原封不动地复用既存类。

§ 对既存类进行演化以得到满足要求的类。演化可以是横向的，也可以是纵向的。横向的演化生成既存类的一个新的版本，而纵向的演化将从既存类导出新类。

§ 重新开始进行开发。

一个新的继承结构将建立两种类：一种是抽象类，它概括了将要表达的概念；另一种是具体类，它要实现这个概念。

③ 求精和维护 ：维护活动是针对应用系统的，但求精过程是针对类和结构的。因为我们利用抽象进行开发，因此，维护活动每时每刻都可能修改这些抽象。随着经验的增长，还可以标识抽象的抽象，使得继承结构通过一般化，增加新的层次。

为便于类的调整，应尽量做到定义与实现分离，实现概念封装和信息隐蔽，使得类具有更大的独立性。在使用一个类或复用一个类时，类与类之间产生一种相互依赖关系。但对一个类的公有界面所做的多次修改不应影响使用它的那些类，在公有界面上增加新的操作不应改变既存的软件。需要谨慎处理的是删除操作或改变操作的特征。

(3) 面向对象软件的开发过程

面向对象软件的开发过程开始于问题论域，经历从问题提出到解决的一系列过程。下面具体说明在过程中的这些步骤。

① 分析阶段 ：分析阶段包括两个步骤：论域分析和应用分析。它们都要标识问题论域中的抽象。在分析中，需要找到特定对象，基于对象的公共特性把它们组合成集合，标识出对这个问题的一个抽象。同时要标识抽象之间的关系，并建立对象之间的消息连接。

§ 论域分析 ：论域分析开发问题论域的模型。论域分析应当在应用分析之前进行，我们在了解问题之前应当对问题敞开思想考虑，考察问题论域内的一个较宽的范围，分析覆盖的范围应比直接要解决的问题更多。

§ 应用分析 ：应用(或系统)分析细化在论域分析阶段所开发出来的信息，并且把注意力集中于当前要解决的问题。因为通过论域分析，分析人员具有了较宽的论域知识，因而能开发出更好的抽象。

② 高层设计 ：在一个纯面向对象环境中，软件体系结构设计与类设计常常是同样的过程，但还是应当把体系结构设计与类的设计分开。在高层设计阶段，设计应用系统的顶层视图。这相当于开发一个代表系统的类，通过建立该类的一个实例并发送一个消息给它来完成系统的“执行”。

③ 类的开发 ：根据高层设计所标识的对各个类的要求和类的规格说明，进行类的开发。因为一个应用系统往往是一个类的继承层次。对这些类的开发是最基本的设计活动。

④ 实例的建立：建立各个对象的实例，实现问题的解决方案。

⑤ 组装测试：按照类与类之间的关系组装一个完整的应用系统的过程中进行的测试。各个类的封装和类测试的完备性可减少组装测试所需要的时间。

⑥ 维护：维护的要求将影响应用和各个类。继承关系可支持对现有应用的扩充，或者加入新的行为，或者改变某些行为的工作方式。

§ 应用系统的维护 ：包括在系统的操作中定位故障、在既存的系统中加入新的行为。应用的维护能够简化对类实例的定位、修改其类的实现、通过改变消息或接收消息的次序来改变应用中特殊对象的角色。新的行为可通过定义新的类和建立实例来实现。

§ 类的维护 ：把类的实现与其规格说明分离可局部化修改的影响。一般情况下，修正问题要求应尽可能不改变类的界面。然而，为了在系统中增加新的行为，偶尔会有改变界面的需求。

3. 面向对象分析(OOA)与模型化

面向对象分析过程分为论域分析和应用分析。论域分析建立大致的系统实现环境，应用分析则根据特定应用的需求进行论域分析。

(1) OOA 分析的基本原则和任务

为建立分析模型，要运用如下的 5 个基本原则：① 建立信息域模型；② 描述功能；③ 表达行为；④ 划分功能、数据、行为模型，揭示更多的细节；⑤ 用早期的模型描述问题的实质，用后期的模型给出实现的细节。这些原则形成 OOA 的基础。

OOA 的目的是定义所有与待解决问题相关的类(包括类的操作和属性、类与类之间的关系以及它们表现出的行为)。为此，OOA 需完成的任务是：

① 软件工程师和用户必须充分沟通，以了解基本的用户需求；

② 必须标识类(即定义其属性和操作)；

③ 必须定义类的层次；

④ 应当表达对象与对象之间的关系(即对象的连接)；

⑤ 必须模型化对象的行为；

⑥ 反复地做任务①－⑤，直到模型建成。

(2) OOA 概述

目前已经衍生许多种 OOA 方法。每种方法都有各自的进行产品或系统分析的过程，有一组可描述过程演进的图形标识，以及能使得软件工程师以一致的方式建立模型的符号体系。现在广泛使用的 OOA 方法有以下几种：

① Booch 方法 ：Booch 方法包含“微开发过程”和“宏开发过程”。微开发过程定义了一组任务，并在宏开发过程的每一步骤中反复使用它们，以维持演进途径。Booch OOA 宏开发过程的任务包括标识类和对象、标识类和对象的语义、定义类与对象间的关系，以及进行一系列求精从而实现分析模型。

② Rumbaugh 方法 ：Rumbaugh 和他的同事提出的对象模型化技术(OMT)用于分析、系统设计和对象级设计。分析活动建立三个模型：对象模型(描述对象、类、层次和关系)，动态模型(描述对象和系统的行为)，功能模型(类似于高层的 DFD，描述穿越系统的信息流)。

③ Coad 和 Yourdon 方法 ：Coad 和 Yourdong 方法常常被认为是最容易学习的 OOA 方法。建模符号相当简单，而且开发分析模型的导引直接明了。其 OOA 过程概述如下：

§ 使用“要找什么”准则标识对象；

§ 定义对象之间的一般化 / 特殊化结构；

§ 定义对象之间的整体 / 部分结构；

§ 标识主题(系统构件的表示)；

§ 定义属性及对象之间的实例连接；

§ 定义服务及对象之间的消息连接。

④ Jacobson 方法 ：也称为 OOSE(面向对象软件工程)。Jacobson 方法与其它方法的不同之处在于他特别强调使用实例(use case)——用以描述用户与系统之间如何交互的场景。Jacobson 方法概述如下：

§ 标识系统的用户和它们的整体责任；

§ 通过定义参与者及其职责、使用实例、对象和关系的初步视图，建立需求模型；

§ 通过标识界面对象、建立界面对象的结构视图、表示对象行为、分离出每个对象的子系统和模型，建立分析模型。

⑤ Wirfs－Brock 方法 ：Wirfs－Brock 方法不明确区分分析和设计任务。从评估客户规格说明到设计完成，是一个连续的过程。与 Wirfs－Brock 分析有关的任务概述如下：

§ 评估客户规格说明；

§ 使用语法分析从规格说明中提取候选类；

§ 将类分组以表示超类；

§ 定义每一个类的职责；

§ 将职责赋予每个类；

§ 标识类之间的关系；

§ 基于职责定义类之间的协作；

§ 建立类的层次表示；

§ 构造系统的协作图。

⑥ 统一的 OOA 方法(UML)

统一的建模语言(UML)已经在企业中广泛使用，它把 Booch、Rumbaugh 和 Jacobson 等各自独立的 OOA 和 OOD 方法中最优秀的特色组合成一个统一的方法。UML 允许软件工程师使用由一组语法的语义的实用的规则支配的符号来表示分析模型。

在 UML 中用 5 种不同的视图来表示一个系统，这些视图从不同的侧面描述系统。每一个视图由一组图形来定义。这些视图概述如下： § 用户模型视图 ：这个视图从用户(在 UML 中叫做参与者)角度来表示系统。它用使用实例(use case)来建立模型，并用它来描述来自终端用户方面的可用的场景。

§ 结构模型视图 ：从系统内部来看数据和功能性。即对静态结构(类、对象和关系)模型化。

§ 行为模型视图 ：这种视图表示了系统动态和行为。它还描述了在用户模型视图和结构模型视图中所描述的各种结构元素之间的交互和协作。

§ 实现模型视图 ：将系统的结构和行为表达成为易于转换为实现的方式。

§ 环境模型视图 ：表示系统实现环境的结构和行为。

通常，UML 分析建模的注意力放在系统的用户模型和结构模型视图，而 UML 设计建模则定位在行为模型、实现模型和环境模型。

(3) 论域分析(Domain Analysis)

论域分析是基于特定应用论域，标识、分析、定义可复用于应用论域内多个项目的公共需求的技术。它的目标是发现和创建一组应用广泛的类，这组类常常超出特定应用的范围，可以复用于其它系统的开发。

论域分析可以被视为软件过程的一种保护伞活动，是与任一软件项目都没有牵连的软件工程活动。论域分析员的工作是设计和建造可复用的构件，供许多工作在类似的但不一定相同的应用项目中的人员使用。

论域分析过程的关键输入／输出参看图 6.10。主要的过程活动有：

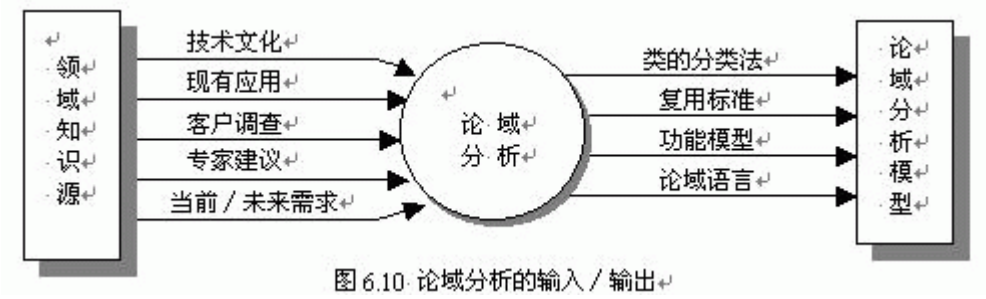


图 6.10 论域分析的输入／输出

① 定义要研究的论域 ：分析员首先隔离感兴趣的业务论域、系统类型或产品分类，再抽取 00 项和非 00 项。其中，00 项包括既存应用的类的规格说明、设计和代码；支持类(GUI 类或数据库存取类)；与论域相关的市售构件库；测试用例等。非 00 项包括方针、步骤、计划、标准和指南；既存的非 00 应用的规格说明、设计和测试信息；度量、市售非 00 软件等。

② 分类从论域抽取的项 ：对所有的项进行归类并定义各个种类的一般定义特征。提出种类的分类模式并定义每个项的命名惯例。适当的时候建立分类的层次。

③ 收集论域中各个应用的有代表性的样例 ：为了完成这个活动，必须保证在问题中的应用具有适合已定义的某些种类的项。

④ 分析样例中的每一个应用 ：分析员接下来要做的事情是：

§ 标识候选的可复用的对象；

§ 指明标识对象为可复用的理由；

§ 定义可复用对象的适合性；

§ 估计在论域中可做到对象复用的应用的百分比；

§ 用名字标识对象并使用配置管理技术控制它们。一旦标识了对象，分析员应当估计一个典型的应用能够使用可复用对象构造的百分比。

此外，论域分析员还应建立一组复用指南，并给出一个例子，说明如何使用论域对象来建立新的应用。

总之，论域分析实际上是一种学习，涉及与应用论域有关的所有知识。论域的边界可能很模糊，很多是凭借经验和实际考虑(如可用资源)。主要思想是想把考虑的论域放宽一些，把相关的概念都标识到，以帮助更好地掌握应用的核心知识。当用户改变他们对系统的需求时，范围广泛的分析可以帮助预测这些变化。

(4) 应用分析

应用分析的依据是在论域分析时建立起来的论域分析模型，并把它用于当前正在建立的应用当中。客户对系统的需求可以当做限制来使用，用它们缩减论域的信息量。就这一点来说，保留的信息受到论域分析视野的影响。论域分析产生的模型并不需要用任何基于计算机系统的程序设计语言来表示，而应用分析阶段产生影响的条件则伴随着某种基于计算机系统的程序设计语言的表示。响应时间需求、用户界面需求和某些特殊的需求，如数据安全等，都在这一层分解抽出。

许多模型识别的要求是针对不止一个应用的。通常我们着重考虑两个方面：应用视图和类视图。必须对每个类的规格说明和操作详细化，还必须对形成应用结构的类之间的相互作用加以表示。

4. 面向对象设计（OOD）

OOA 和 OOD 之间有密切的衔接关系，从 OOA 到 OOD 是一个逐渐扩充模型的过程。分析处理以问题为中心，可以不考虑任何与特定计算机有关的问题，而 OOD 则把我们带进了面向计算机的“实地”开发活动中去。通常，OOD 分为两个阶段，即高层设计和低层设计。高层设计建立应用的体系结构。低层设计集中于类的详细设计。

(1) 高层设计

高层设计阶段开发软件的体系结构，构造软件的总体模型。在这个阶段，标识在计算机环境中进行问题解决工作所需要的概念，并增加了一批需要的类。这些类包括那些可使应用软件与系统的外部世界交互的类。此阶段的输出是适合应用软件要求的类、类间的关系、应用的子系统视图规格说明。通常，利用面向对象设计得到的系统框架如图 6.11 所示。

① 高层设计模型

一个典型的高层设计模型即客户－服务器模型，它构造起应用软件的总体模型，这个模型导出的体系结构既可在过程性系统中使用，又可在面向对象的系统中使用。

客户－服务器模型的想法是让系统的一个部分(服务器子系统)提供一组服务给系统的另一个部分(客户子系统)。请求服务的对象都归于客户子系统，而接受请

求提供服务的部分就是服务器。X－window 系统在用户界面设计时就使用了客户—服务器方法来构造它的图形交互界面。一个应用的核心“引擎”是服务器子系统的客户，如用户界面、打印服务器、数据库服务器等。

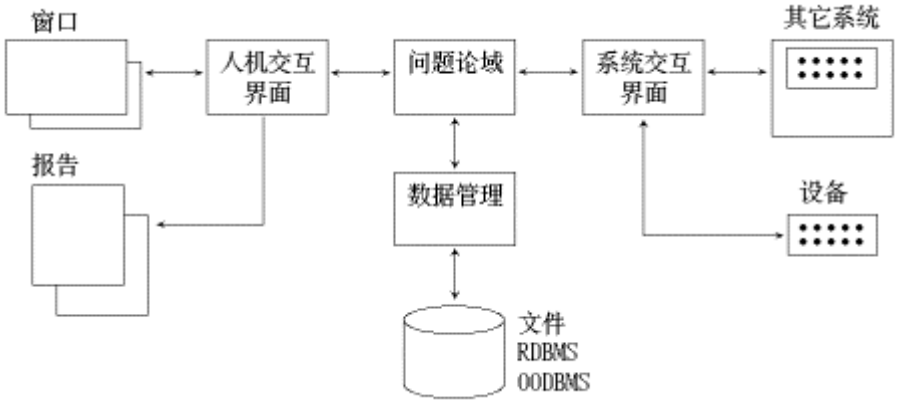


图 6.11 OOD 设计导出的体系结构

在 Smalltalk 中使用的软件体系结构是模型／视图／控制器(MVC，Model／View／Controller)，参看图 6.12。在这个结构中，模型是软件中的应用论域的各种对象，它们的操作独立于用户界面；视图则管理用户界面的输出；而控制器处理软件的输入。输入事件给出要发送给模型的消息。一旦模型改变了它的状态，就立即通过关联机制通知视图，让视图刷新显示。这个关联机制定义了在模型与各个视图之间的关系，它允许模型的运行独立于与它相关联的视图。类似地，控制器在输入事件发生时将对视图及模型进行控制与调度。

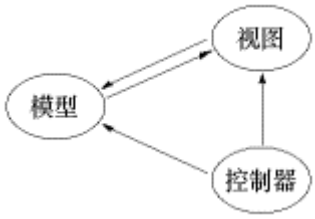


图 6.12 MVC 框架结构

此外，在软件体系结构中还使用了许多其它配置。类的配置表示的多是子系统的体系结构而不是系统的总体结构。它们是一些抽象类的集合，可以定义这些抽象类的新的派生类，并把它们实例化，以实现所要求的特定行为，籍此构造要求的应用软件。对于 MVC 来说，可以通过开发模型的一个派生类，履行与应用相关联的处理，建立这个应用软件。用户界面通过定义视图和控制器的派生类来建立，这些派生类中许多是可复用的既存类，像按钮和对话框等在多数 Smalltalk 系统中已经存在。这样就导致了新子系统的建立。

② 高层设计的规则

§ 最小化各构件间的通信：在子系统的各个高层构件之间的通信量应当达到最小。一个用户界面应当能够自行处理交互、错误改正和硬件控制，而不需打扰主应用。

§ 隐藏复杂性：子系统应当把那些成组的类打包，形成高度的内聚。 § 逻辑功能分组：虽然输入和输出设备可能相互间不通信，但逻辑上把它们归组到一个处理输入 / 输出的子系统中。这样比较容易识别并定位问题论域中的事件。

类与通过概念封装的子系统十分类似。事实上，每个子系统都可以被当做一个类来实现，这个类聚集它的构件，提供了一组操作。类和子系统的结构是正交的，一个单个类的实例可能是不止一个子系统的一部分。

高层设计阶段增加了一批必要的类，主要包括了那些可使应用软件与系统的外部世界交互的类。这些交互则包括与其它软件系统(如数据库管理系统、鼠标和键盘)的界面，与使用来进行数据收集或者负责控制的硬件设备的界面等。

高层设计可以表征为标识和定义模块的过程。但这种模块可以是单个的类，还可以是由一些类组合成的子系统。定义过程是职责驱动的。

高层设计和类设计这两个阶段是相对封闭的。在这种情况下，应用软件中的每一个事物都是一个对象，包括应用软件自身在内！根据这个思想，这两个阶段又是连接的。应用软件的设计是大类的设计，这种类设计考察应用软件所期望的每一个行为，并利用这些行为形成应用类的界面。

(2) 类设计的目标和方针

类设计的第一步是标识应用所需的概念。应用分析过程包括了对问题论域所需的类的模型化；但在最终实现应用时不只有这些类，还需要追加一些类。在类设计的过程中应当做这些工作。类设计的主要目标如下：

① 单一概念的模型 ：在分析与高层设计阶段，常常需要使用多个类来表示一个“概念”。一般人们在使用面向对象方法开发软件时，常常把一个概念进行分解，用一组类来表示这个概念。当然，也可以只用一个独立的类来表示一个概念。

② 可复用的“插接相容性”构件 ：我们希望所开发的构件可以在未来的应用中使用。因此，需要一些附加特性。例如，在相关的类的集合中界面的标准化，在一个集合内部的类的“插接相容性”等。

③ 可靠的构件 ：应用软件必须是可靠的(健壮的和正确定义的)软件。而这种可靠性与它的构件有关。每个构件必须经过充分的测试。但由于成本关系，往往测试不够完备。然而，如果我们要建立可复用的类，则通过测试确保构件的可靠性是绝对必要的。

④ 可集成的构件 ：我们希望把类的实例用到其它类的开发和应用中，这要求类的界面应当尽可能小，一个类所需要的数据和操作都定义在类定义中。因此，类

的设计应当尽量减少命名冲突。面向对象语言的消息语法可通过鉴别带有实例名的操作名来减少可能的命名冲突。

类结构提供的封装使得把概念集成到应用的工作变得很容易。封装特性保证了把一个概念的所有细节都组合在一个界面下，而信息隐蔽则保证了实现级的名字将不会其它类的名字互相干扰。

我们讨论的方针是类的模块设计的方针，还要给出类设计质量的度量。

① 信息隐蔽 ：软件设计通过信息隐蔽可增强抽象，并可保护类的存储表示不被抽象数据类型实例的用户直接存取。对其表示的唯一存取途径只能是界面。

② 消息限制 ：类的设计者应当为类的命令设计一个明确的界面，该类实例的用户应当只使用界面提供的操作。

③ 狭窄界面 ：不是所有的操作都是公共的。只有对其它类的实例是必要的操作才放到界面上，其它操作应是隐蔽实现的部分。

④ 强内聚 ：模块内部各个部分之间应有较强的关系，它们不能分别标识。

⑤ 弱耦合 ：一个单独模块应尽量不依赖于其它模块。如果在类 A 的实例中建立了类 B 的实例，或者如果类 A 的操作需要类 B 的实例做为参数，或者如果类 A 是类 B 的一个派生类，则称类 A“依赖于”类 B。一个类应当尽可能少地依赖于其它类。

耦合程度部分依赖于所使用的分解方法。类 A 之所以依赖于类 B，是因为类 A 要求类 B 提供服务。这个依赖性可通过复制类 A 中的类 B 的功能来消除。但代码的复制减少了系统的灵活性并增加了维护的困难。继承结构损害了弱耦合的概念。因为在建立一般化-特殊化关系的时候，继承引入了依赖。

⑥ 显式信息传递 ：除了依赖于最少的类外，还应该明确在这些类之间的信息流。在类之间全局变量的共享隐含了信息的传递，并且是一种依赖形式。因此，两个类之间的交互应当仅涉及显式信息传递。显式信息传递是通过参数表来完成的。

⑦ 派生类当做派生类型 ：继承结构的使用是面向对象开发方法的一大特色。每个派生类应该当做基类的特殊化来开发，而基类所具有的公共界面成为派生类的共有界面的一个子集。C++允许设计者选择类的基类是共有的或私有的。如果基类是共有的，则其共有界面将成为新的派生类的共有界面部分，这表明基类的行为成为派生类的行为部分。这类似于类型与派生类型之间的关系。如果基类是私有的，它的行为将不是继承类的公共行为部分而是实现部分。它的提出是为了提供实现新类的服务。

⑧ 抽象类 ：某些语言提供了一个类，用它做为继承结构的开始点，所有用户定义的类都直接或间接以这个类为基类。Smalltalk 提供了一个类 Object 做为所有类的继承树的根，而 C++则支持多重继承结构。每一种结构都包含了一组类，

它们是(或应该是)某种概念的特殊化。这个概念应抽象地由结构的根类来表示。因此，每个继承结构的根类应当是目标概念的一个抽象模型。这个抽象模型生成一个类，它不用于产生实例。它定义了一个最小的共有界面，许多派生类可以加到这个界面上以给出概念的一个特定视图。

(3) 通过复用设计类

利用既存类来设计类，有 4 种方式：选择，分解，配置和演变。这是面向对象技术的一个重要优点。许多类的设计都是基于既存类的复用。

① 选择 ：设计类最简单的方法是从既存构件中简单地选择合乎需要的构件。这就是开发软件库的目的。一个 00 开发环境应提供常用构件库，大多数语言环境都带有一个原始构件库(如整数、实数和字符)，它是基础层。任一基本构件库(如“基本数据结构”构件)都应建立在这些原始层上。这些都是些一般的和可复用的类。这个层还包括一组提供其它应用论域服务的一般类，如窗口系统和图形图元。表 6.1 显示了建立在这些层上面的特定域的库。最低层的论域库包括了应用论域的基础概念并支持广泛的应用开发。特定项目和特定组的库包括一些论域库，它包含为相应层所定义的信息。

表 6.1 一个面向对象构件库的层次

特定组的构件	—	一个小组为他们自己组内所有成员使用而开发
特定项目的构件	—	一个小组为某一个项目而开发
特定问题论域的构件	—	购自某一个特定论域的软件销售商
一般构件	—	购自专门提供构件的销售商
特定语言原操作	—	购自一个编译器的销售商

② 分解 ：最初标识的“类”常常是几个概念的组合。在设计时，可能会发现所标识的操作落在分散的几个概念中，或者会发现，数据属性被分开放到模型中拆散概念形成的几个组内。这样我们必须把一个类分成几个类，希望新标识的类容易实现，或者它们已经存在。

③ 配置 ：在设计类时，可能会要求由既存类的实例提供类的某些特性。通过把相应类的实例声明为新类的属性来配置新类。例如，一种仿真服务器可能要求使用一个计时器来跟踪服务时间。设计者不必开发在这个行为中所需的数据和操作，而是应当找到计时器类，并在服务器类的定义中声明它。

④ 演变 ：要开发的新类可能与一个既存类非常类似，但不完全相同。此时，不适宜采用“选择”操作，但可以从一个既存类演变成一个新类，可以利用继承机制来表示一般化-特殊化的关系。特殊化处理有三种可能的方式。
§ 由既存类建立子类 ：现要建立一个新类“起重车”。它的许多属性和服务都在既存类“汽车”中。关系如图 6.13 所示。新类是既存类的特殊情形。这时直接让“起重车”类作为“汽车”类的子类即可。



图 6.13 建立子类

② 建立继承层次由既存类建立新类 ：现要增加一个新类“拖拉机”。它的属性与服务有的与“汽车”类相同，有的与“汽车”类不同。关系如图 6.14 所示。这时，调整继承结构。建立一个新的一般的“车辆”类，把“拖拉机”与“汽车”类的共性放到“车辆”类中，“拖拉机”与“汽车”类都成为“车辆”类的子类。“车辆”是抽象类，相关操作到子类“汽车”类去找。

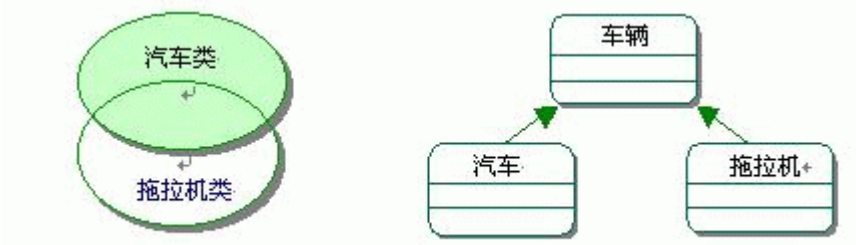


图 6.14 调整继承结构

③ 建立既存类的父类 ：另一种情形是想在既存类的基础上加入新类，使得新类成为既存类的一般类。例如，已经存在“三角形”类，“四边形”类，想加入一个“多边形”类，并使之成为“三角形”和“四边形”类的一般类。继承结构如图 6.15 所示。从这个“多边形”类又可派生出新的类，如“六边形”类。

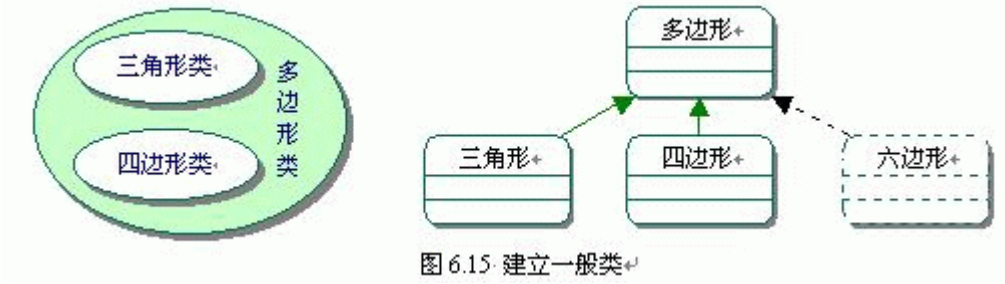


图 6.15 建立一般类

后两种涉及既存类的修改。在这两种情况下，既存类中定义的操作或数据被移到新类中。如果遵循信息隐蔽和数据抽象的原理，这种移动应不影响已有的使用这些类的应用。类的界面保持一致，虽然某些操作是通过继承而不是通过类的定义伸到这个类的。

(4) 类设计方法

通常，类中的实例具有相同的属性和操作，应当建立一个机制来表示类中实例的数据表示、操作定义和引用过程。这时，类的设计是由数据模型化、功能定义和 ADT 定义混合而成的。类是某些概念的一个数据模型，类的属性就是模型中的数

据域，类的操作就是数据模型允许的操作。要明确规定它们两个谁先确定是不可能的，两个处理是互补的。

类的标识有主动和被动之分。被动类是数据为中心的，它们是根据系统的其它对象发送来的消息而修改其封装数据的；主动类则提供许多系统必须履行的基本操作。与被动类的实例(被动对象)一样，主动类的实例(主动对象)接收消息，但这些对象是负责发送追加消息和控制某些应用部分的。在窗口环境，一个窗口是一个被动对象，它基于发送给窗口的消息来显示某些内容。窗口管理器是一个主动对象，它担负着各种在它控制的窗口上的操作。

在被动类与主动类的设计之间不存在明显的差别。在设计主动类时，需要优先确定数据模型，稍后再确定操作；在设计被动类时，把类提供的服务翻译成操作。在标识了服务之后再设计为支持服务所需要的数据。许多类都是这两个极端的混合。

类中对象的组成包括了私有数据结构、共享界面操作和私有操作。而消息则通过界面，执行控制和过程性命令。因此，要分别讨论它们的实现。

类的设计描述包括两部分：

① 协议描述 ：协议描述定义了每个类可以接收的消息，建立一个类的界面。协议描述由一组消息及对每个消息的相应注释组成。

② 实现描述 ：实现描述说明了每个操作的实现细节，这些操作应包含在类的消息中。实现描述由以下信息构成：

§ 类名和对一个类引用的规格说明

§ 私有数据结构的规格说明，包括数据项和其类型的指示

§ 每个操作的过程描述

实现描述必须包含充足的信息，以提供在协议描述中所描述的所有消息的适当处理。由一个类所提供服务的用户必须熟悉执行服务的协议，即定义“什么”被描述；而服务的提供者（对象类本身）必须关心：服务如何提供给用户，即实现细节的封装问题。

5. 对象模型技术

Rumbaugh 等人提出对象模型技术(OMT)，它把分析时收集的信息构造在三类模型中，即对象模型、功能模型和动态模型。图 6.16 给出了这三个模型的建立次序。从功能模型回到对象模型的箭头表明，这个模型化的过程是一个迭代的过程。每一次迭代都将对这三个模型做进一步的检验、细化和充实。

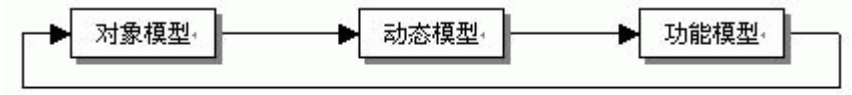


图 6.16 对象模型、功能模型和动态模型的建立次序

(1) 对象模型

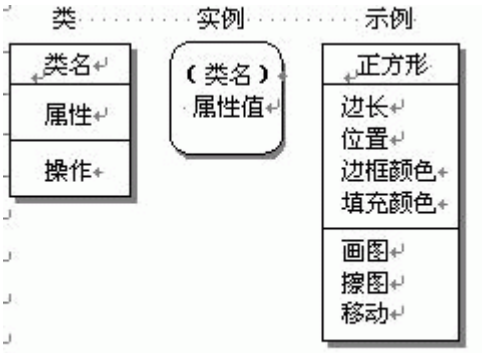


图 6.17 类与对象的表示方法

对象模型是三个模型中最关键的模型，它的作用是描述系统的静态结构，包括构成系统的类和对象，它们的属性和操作，以及它们之间的关系。事实上，这个模型可以看作扩充的实体－关系模型(E－R)。图 6.17 给出了在对象模型中用以表示类和对象的图形符号。在 OMT 中，类与类之间的关系叫做关联。关联代表一组存在于两个或多个对象之间的、具有相同结构和含义的具体连接。关联可以是物理的，也可以是逻辑的。

图 6.18 给出与关联有关的几个特殊图形符号。

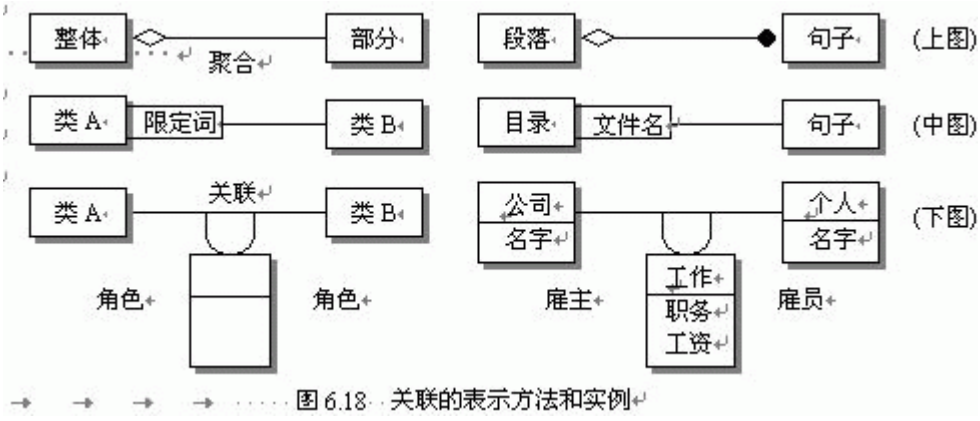


图 6.18 关联的表示方法和实例

上图是聚合，代表整体与部分关系，右边的例子说明“(文章中的)段落是由多个句子组成”。句子一端的实心圆 ● 表示“多个”，空心圆 ○ 表示“0 个”，不加圆表示“1 个”。中图是限定，对关联的含义做某种约束。右边的例子表明，附加的限定词在说明“一个目录包含多个文件”的基础上，更明确地指明“每一个文件都可由目录中的文件名属性来唯一地标识”。下图是角色，由于多数关联具有两个端点，因而涉及到两个角色。右边的例子表明，在公司与个人的关联中，

公司的角色是雇主，个人的角色是雇员。 此外，还可用连接属性进一步说明对象之间的连接。再看图中例子，由于一个人在成为公司的雇员时才有工资和职务。因此，工资和职务不是个人属性，而是公司与个人之间的连接属性。OMT 的提出者指出，这种情况常常意味着有必要定义一个新类来取而代之。

OMT 还定义了一般化与特殊化关系(也称为继承性)。一般化与特殊化关系通常包含一个(如果允许多重继承性，也可以是几个)基类和几个派生类。参看图 6.19。基类表示了一个较为一般、普遍的概念，而每个派生类则是它的某个特殊形态。因而派生类除了自然地继承基类所具有的属性和操作外，还具有反映自身特点的属性和操作。

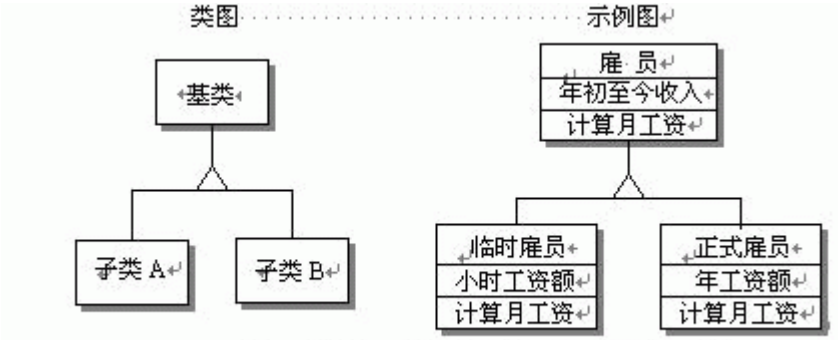
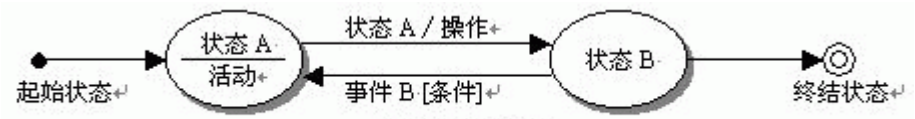


图 6.19 一般化与特殊化关系(继承性)的表示方法和示例

(2) 动态模型

要想对一个系统了解得比较清楚，首先应考察它的静态结构，即在某一时刻它的对象和这些对象之间相互关系的结构；然后应考察在任何时刻对对象及其关系的改变。系统的这些涉及时序和改变的状况，用动态模型来描述。动态模型着重于系统的控制逻辑。它包括两个图，一是状态图，一是事件追踪图。

① 状态图 ：状态图是一个状态和事件的网络，侧重于描述每一类对象的动态行为。在状态图中，状态是对某一时刻中属性特征的概括。而状态迁移则表示这一类对象在何时对系统内外发生的哪些事件做出何种响应。图 6.20 给出了状态图的表示方法。图中的椭圆表示状态，状态之间的箭头表示从一个状态到另一个状态的迁移，附加在箭头上的短语说明触发此状态迁移的事件。在图 6.20 中，“事件 A”是一个单纯的事件，而“事件 B[条件]”是一个有条件的事件，在给定条件满足时才起作用。OMT 区分两种不同的行为，即操作和活动。操作是一个伴随状态迁移的瞬时发生的行为，与触发事件一起表示在有关的状态迁移之上。活动则是发生在某个状态中的行为，往往需要一定的时间来完成，因此与状态名一起出现在有关的状态之中。状态图中所有这些成份都可以根据具体要求而予以取舍。



6.20 状态图的表示方法

n 对一个事件的响应依赖于接收它的对象的状态，它可以包括状态的改变、发送另一个事件给原来的发送者或第三个对象。对于一给定类的事件、状态及状态的迁移，可以抽象并表示成一个状态图。动态模型由多个状态图组成，对于每一个具有重要动态行为的类都有一个状态图，从而表明整个系统活动的模式。各个状态图并发地执行，并可以独立地改变状态。对于各种类的状态图可以通过共享事件组合到一个动态模型中。

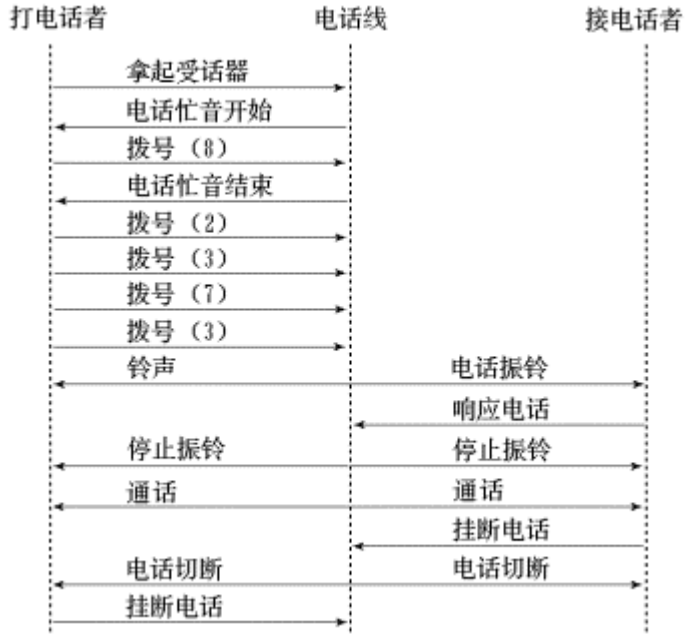
② 事件 ：一个事件发生在某一时刻。一个事件在逻辑上可能领先于另一个事件，或者两者没有关系。两个没有因果关系的事件叫做并发的，它们之间互相不影响。在模型化一个系统时，一般不考虑并发事件的次序，因为它们可能以任一次序发生。一个分布系统的模型必须包括并发事件和活动。事件都是单独发生的，我们把它们纳入事件类中，并给每个事件一个名字，以指明共同结构和行为。对于所有的事件来说，事件发生的时间是隐蔽的。 事件从一个对象向另一个对象传送信息。有些事件类可能传送简单的信号“要发生某件事”，而有些事件类则可能传送数据值。由事件传送的数据值叫做属性，属性可以在事件类名之后用括号列出。

③ 事件追踪图 ：事件追踪图侧重于说明发生于系统执行过程中的一个特定“场景”(即脚本)，是完成系统某个功能的一个事件序列。场景通常起始于一个系统外部的输入事件，结束于一个系统外部的输出事件，它可以包括发生在这个期间的系统所有的内部事件，也可以只包括那些撞到的或由系统中某些对象生成的事件。脚本可以是系统执行的历史记录，还可以是执行一个拟议中系统的设想实验。下面是使用电话的一个脚本。这个脚本只包含影响电话的事件。

[1] 打电话者拿起电话受话器 → → [10] 铃声在打电话者的电话上传出
[2] 电话忙音开始 → → → → [11] 接电话者回答
[3] 打电话者拨数字 (8) → → → [12] 接电话者的电话停止振铃
[4] 电话忙音结束 → → → → [13] 铃声在打电话者的电话中消失
[5] 打电话者拨数字 (2) → → → [14] 通电话
[6] 打电话者拨数字 (3) → → → [15] 接电话者挂断电话
[7] 打电话者拨数字 (7) → → → [16] 电话切断 →
[8] 打电话者拨数字 (3) → → → [17] 打电话者挂断电话
[9] 接电话者的电话开始振铃

每个事件从一个对象向另一个对象传送信息。在写出这个场景后，下一步就是要标识每个事件的发送者对象和接收者对象。各种有关事件的序列关系以及由此而表现出来的对象之间的交互作用可以通过事件追踪图表达。在这个图上，竖线表示对象，带有箭头的横线表示事件，箭头从发送者对象指向接收者对象。时间自上向下延续，与间隔的空间无关。这只是事件序列的表示，没有精确的时序。图

6.21 就是一个打电话的事件追踪图。此外，并发事件也可以发送。例如，电话线并发地发送事件给打电话者与接电话者。



概括地讲，状态图叙述一个对象的个体行为，而事件追踪图则给出多个对象所表现出来的集体行为。它们从不同侧面来说明同一系统的行为，因此必然存在着密切的内在关系。例如，一个事件追踪图指出某一对象在接受一个事件之后发出另一事件，同一行为在此对象的状态图中也应当有所表示。另外，由于状态图的复杂程度会随着应用问题的增大而呈指数型迅速增加，这使得分析和验证十分困难。针对这个问题，OMT 允许将一个复杂状态分解为一些比较简单的子状态，以前者对系统或系统的某一部分做高度的抽象和概括，以后者来提供在前者中省略掉的有关细节。这样，不仅保证了对系统行为的充分说明，而且有效地限制了状态图本身的复杂性。

(3) 功能模型

功能模型着重于系统内部数据的传送和处理。它是模型化三角架的第三只脚。功能模型定义“做什么”，动态模型定义“何时做”，对象模型定义“对谁做”。

功能模型表明，通过计算，从输入数据能得到什么样的输出数据，不考虑参加计算的数据按什么时序执行。功能模型由多个数据流图组成，它们指明从外部输入，通过操作和内部存储，直到外部输出，这整个的数据流情况。功能模型还包括了对象模型内部数据间的限制。数据流图不指出控制或对象的结构信息，它们包含在动态模型和对象模型中。

图 6.22 给出了数据流图的表示方法。图中的实线表示数据流，虚线表示控制流，圆框代表处理数据的过程，矩形框表示产生与接收数据的对象，平行线框表示数据存储区。熟悉数据流图的人可以发现，它基本上就是传统的数据流图加上控制

流。但 OMT 指出，虽然控制流有时是有用的，但它重复了动态模型中的有关部分，因而应尽量少用。

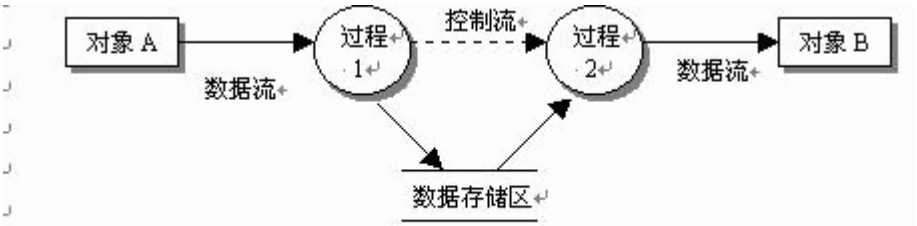


图 6.22 数据流图的表示方法

功能模型中所有的数据流图往往形成一个层次结构。在这个层次结构中，一个数据流图中的过程可以由下一层的数据流图做进一步的说明。一般来讲，高层的过程相应于作用在聚合对象上的操作，而低层的过程则代表作用于一个简单对象上的操作。

(4) 基于三个模型的分析过程 OMT 强调对“现实”的模型化。这里，“现实”是指我们所面对的实际应用问题。即就是说，分析模型是对实际问题的抽象。分析模型中的对象代表实际问题中的对象，分析模型中对象之间的关系反映了实际问题中对象之间的关系。与此同时，OMT 还对具体做法给出很多建议。比如，如何识别类及类与类之间的关系，何时考虑属性和操作，怎样利用继承对对象模型做进一步的细化等。此外，OMT 还建议通过识别触发系统内部活动的事件，以及系统对这些事件的反应来建立动态模型；通过分析输入和输出数据(这些通常是伴随事件的有关参数)的相关性，以及有关的中间结果来建立功能模型。

6. Coad 与 Yourdon 的 OOA 方法

我们采用 Coad 与 Yourdon 提出的面向对象分析方法来讨论分析的过程。这种方法是在信息模型化技术，面向对象程序设计语言及知识库系统的基础上发展起来的。

(1) 面向对象的分析的考虑

Coad 与 Yourdon 认为面向对象分析(OOA)的主要考虑在于与一个特定应用有关的对象以及对象与对象之间在结构与相互作用上的关系。OOA 有两个任务。

§ 形式地说明我们所面对的应用问题，最终成为软件系统基本构成的对象，还有系统所必须遵从的，由应用环境所决定的规则和约束。
§ 明确地规定构成系统的对象如何协同合作，完成指定的功能。这种协同在模型中是以一组消息连接来表示的，它们承担了各个对象之间的通信。 通过 OOA 建立的系统模型是以概念为中心的，因此称为概念模型。这样的模型由一组相关的类组成。OOA 可以采用自顶向下的方法，逐层分解建立系统模型，也可以自底向上地从已经定义的基本类出发，逐步构造新的类。软件规格说明就是基于这样的概念模型形成的，以模型描述为基本部分，在加上接口要求、性能限制等其它方面的要求说明。

构造和评审 OOA 概念模型的顺序和由五个层次组成。这五个层次不是构成软件系统的层次，而是分析过程中的层次，也可以说是问题的不同侧面。每个层次的工作都为系统的规格说明增加了一个组成部分。当五个层次的工作全部完成时，OOA 的任务也就完成了。这五个层次是：类与对象、属性、服务、结构和主题。

图 6.23 给出了这五个层次，以及每个层次中涉及到的主要概念和相应的图形表示。

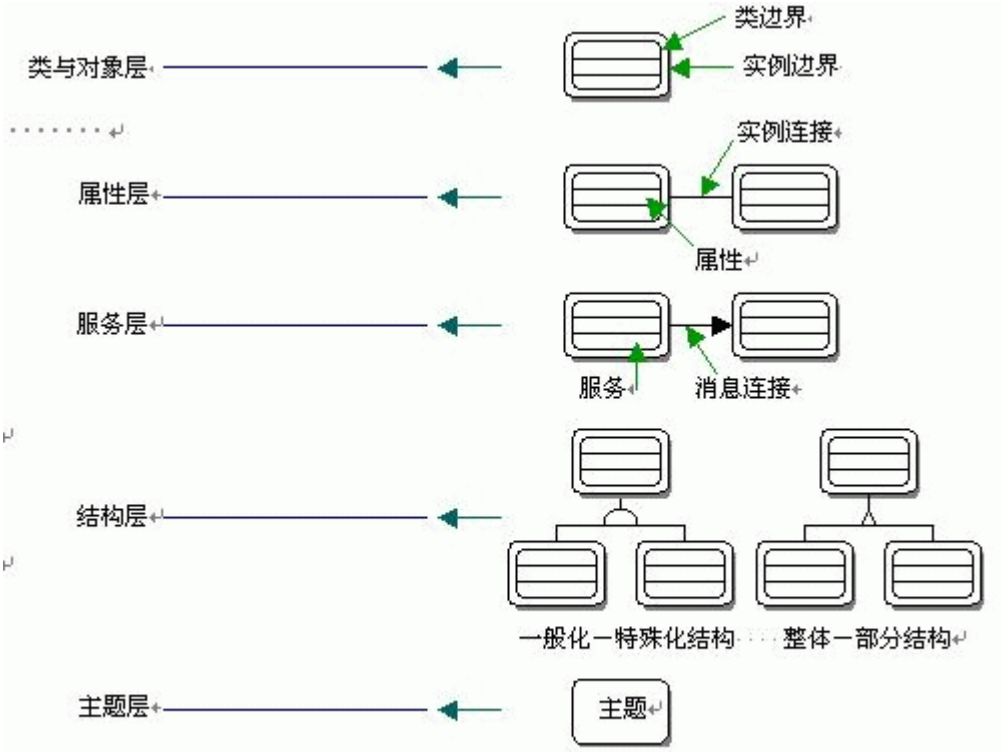


图 6.23 面向对象分析的概念模型

(2) 标识对象和类

OOA 的第一个层次是识别类和对象。类和对象是对与应用有关的概念的抽象。不仅是说明应用问题的重要手段，同时也是构成软件系统的基本元素。这一层的工作是整个分析模型的基础。

① 如何找出对象和类

对于一个给定的问题论域，找出一个适当的对象集合才能确保可复用性和可扩充性，并能借助面向对象的开发模式，提高软件开发的质量和生产率。

在做问题识别时，系统分析员应按以下五个步骤找出描述应用论域的对象和类：

§ 亲自到现场了解在相同应用论域中类似系统的运行情况；

§ 注意听取该应用论域的专家对问题解决相关问题的解说，并进行讨论；

§ 调查以往是否有相同或类似应用论域的面向对象分析的结果。若有，看是否有可复用对象和类。

§ 详细阅读客户提交的问题陈述，并阅读与该应用论域相关的资料或书籍。同时，将在客户的问题陈述中出现的所有名词开列出来。

§ 建立系统原型，交给客户试用，根据客户的意见和要求，修改系统规格说明。

② 基于语言的信息分析(Linguistic－based Information Analysis，简称 LIA)
对象是与应用论域的有关概念紧密结合在一起的。概念的处理主要是基于自然语言：书面语或口语。将语言规则应用到软件系统分析中的处理方法称为基于语言的信息分析(LIA)。LIA 的目的是标识出问题论域的所有概念及这些概念之间的关系。有两种十分有效的 LIA 技术：短语频率分析和矩阵分析。
§ 短语频率分析 ：短语频率分析搜索选定的问题陈述，标识可以表示问题论域概念的术语。PFA 的优点在于能广泛地标识问题论域的概念集合，并对它们进行评估，判定它们中哪些与目标软件无关，避免因某些主观原因，而导致误选、遗漏或产生冗余。
§ 矩阵分析 ：矩阵的行和列是应用论域的概念，这些概念通常可以产生所标识对象的初始集合。矩阵中的元素表示了相对应的行与列上的概念之间的关联关系。初始矩阵可以从问题陈述中机械地构造出来，分析员逐个元素地评审和讨论这个矩阵，标识出应用论域的关系(事务规则)。

③ 三视图模型 3VM（3-View Modeling）

在做面向对象分析时，还用到 3 种非常有用的传统系统分析工具：数据流图、实体－关系图以及状态－迁移图，它们涉及系统的 3 个不同的、相互独立的方面（处理过程、数据和控制），称为三视图模型 3VM（3-View Modeling）。

§ 实体－关系图(简称 ERD)
在 ERD 中的实体很有可能成为对象，实体的属性可表示成最终要由对象进行存储的数据，实体之间的关系有可能将建立“关联对象”。此外，表示关系(连接)的基数，以及因果条件可能成为维持这些关系的服务。所以 ERD 是 OOA 的一个有力的工具。ERD 的问题是：标识的实体可能与应用论域概念无关，且对于那些不存储数据的对象显得无能为力。
§ 数据流模型
有两种形式的数据流模型：上下文图和分层的数据流图。上下文图确定一个全局的系统边界，描述系统和外部实体的交互，这些外部实体应是候选的对象。对象集合必须阐明外部实体的数据流是如何被接收、处理及生成的。分层的数据流图描述系统的功能是如何分解成为一些基本单元的，这些基本单元最终将成为系统的基本处理规格说明 (Primitive Process Specifications, 简称 PPSs)，以 PPSs 描述对象的方法或服务。此外，还可用判定表或场景描述系统的功能。
§ 状态－迁移模型
有两种形式的状态－迁移模型：事件－响应模型和状态－迁移图。任何一个系统中的对象必须是事件的识别者或事件的响应者。如果一个对象既不能识别事件的发生，也不能响应任一事件，它就不是属于这个系统的对象。因此，事件－响应

模型必须标识出系统必须识别的所发生的每一个事件，以及系统必须作出的预期响应的事件，还必须标识一系列的识别事件的对象、产生响应的对象。状态－迁移图则可用来标识事件识别对象和对事件产生响应的对象，标识保存状态信息的属性。

② 选择类的原则

§ 对象应该具有记忆其自身状态的能力。而且对象的属性应当是系统所关心的，或是系统正常运行所必需的。

§ 对象应当具有有意义的服务（操作），可用以修改对象本身的状态（属性值）。而且对象可以利用其服务为系统中的其它对象提供外部服务。如果类不需提供任何服务，则此类或对象可不必包含在分析模型中。

§ 对象应当具有多个有意义的属性。仅有一个属性的对象最好表示为其他对象的属性。

§ 为对象定义的属性应适合于对象的所有实例。如果对象的某一个实例不具备某属性，则意味着应用论域中存在尚未发现的继承关系。应该利用继承关系将原来的对象和特殊的实例区分为两个对象。

§ 为对象定义的有关服务应适合于对象的所有实例。

§ 一个类中应当有一个以上的实例：如果一个类中只要一个实例或对象，那么有可能这个类没有存在的必要。但如果一个类虽然只有一个实例，但它反映的应用论域的某一种概念，那么它必须作为一个类存在。

§ 对象应是软件分析模型的必要成分，与设计和实现方法无关。

§ 有时为了提高执行速度，需要增加一些类或对象或属性，用以保存另一些类或对象产生的暂时结果，以避免重复计算。但这应是设计阶段考虑的问题，不应是分析阶段做的事。 在面向对象的分析活动中，对对象的识别和筛选取决于应用问题及其背景。也取决于分析员的主观思维。

(3) 标识结构

OOA 的下一个层次是结构层，用于标识了对象之间的组装和继承关系。在 OOA 中，标识结构是处理 OOA 模型复杂性的机制之一。典型的结构有两种：一般化－特殊化结构和整体－部分结构，如图 6.24 和 6.25 所示。

整体－部分结构也叫做组装结构，表示聚合关系，即由属于不同类的成员聚合而形成新的类，它用符号△表示有向性，在△的上面是一个整体对象，下面是部分对象。

一般化－特殊化结构也叫做分类结构。其中，特殊化类是一般化类的派生类，一般化类是特殊化类的基类。分类结构具有继承性，一般化类和对象的属性和服务一旦被识别，即可在特殊化类和对象中使用。采用继承来显式地表达属性和服务的公共部分，这可以实现在分类结构中恰如其分地分配属性和服务。将共同的属性放在上层，而将特有的属性放在下层；将共同的服务放在上层，而将特有的服务放在下层。分类结构还可以形成层次或网络，以描述复杂的特殊化类，有效地表示公共部分。

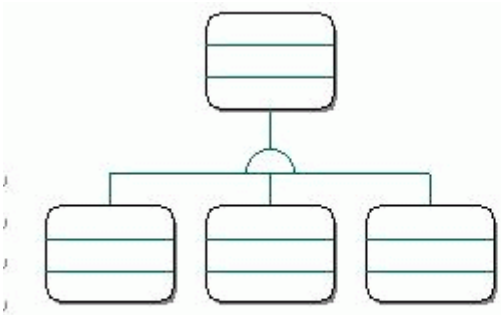


图 6.24 一般化－特殊化结构的例子

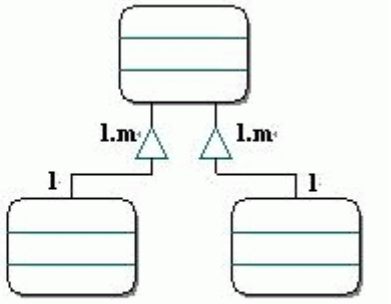


图 6.25 整体－部分结构的例子

继承适合于结构、表格和定义，不适用于值。例如，“鸡”是“鸟”的特殊化，它们之间共享了许多属性和服务。如“翅膀”。“鸡”和“鸟”可以共享“翅膀”的定义，但不能共享“翅膀”的特定值。

从整体的视点来看，一个整体－部分结构可看作一个“has a”结构。例如，Vehicle has a Engine 。其中，Vehicle 是整体对象，Engine 是局部对象。

整体－部分结构不是类与类之间的对应关联，而是类的实例与实例之间的对应关联。一个整体对象可以有不同种类的部分对象，且可以有多个部分对象。因此，在整体－部分连线的两端应标注一个数量(Amount)或范围(Range)，以表明一个整体对象可以拥有的部分对象的数目。通常，这种结构表明以下几种关联：

§ 总体－部分(Assembly－Parts)关联，如飞机－发动机之间的关系。
§ 包容－内含(Container－Content)关联，如飞机－飞行员之间的关系。
§ 收集－成员(Collection－Members)关联，如机构－职员之间的关系。

从特殊化的视点来看，一个一般化－特殊化结构可以看作是“is a”或“is a kind of”结构。例如，a Truck Vehicle is a Vehicle 或 a Truck Vehicle is a kind of Vehicle。

一般化－特殊化结构形成的继承关系可能是单继承，也可能是多继承。它们形成一个层次结构。在单继承的层次结构中，派生类之间可能会出现一些冗余信息；而多继承的层次结构常常会导致较多的派生类，增加了模块复杂性，但能够很好地标识共同性。

(4) 标识属性

下一个层次为属性层，属性层包括对象属性和对象之间的关系(实例连接)。对象所保存的信息称为它的属性，通常把它们封装在对象内部。类的属性所描述的是状态信息，每个实例的属性值表达了该实例的具体状态值。实例连接可以看成是一种事务规则或应用论域限制，表达了一个类的实例与其它类的实例的可能对应关系。

① 标识属性的策略

分析人员应从问题陈述中搞清：哪些性质在当前问题的背景下完全刻画了被标识的某个对象？通常，属性对应于带定语的名词。如“文件的密码”、“学生的出生年月”等等。属性在问题陈述中不一定有完整的显式的描述，要识别出所关心的潜在属性，需要对应用论域有深刻的理解。

§ 每个对象至少应含有一个属性，使得对象的实例能够被唯一地标识。
§ 必须仔细地定义属性的取值。属性的取值必须能应用于对象类中的每一个实例。其取值不能为“不适用”。
§ 出现在一般化－特殊化关系中的对象所继承的属性必须与一般化－特殊化关系一致。子对象不能继承那些不是为该子对象定义的属性。所继承的属性必须在应用论域中有意义。
§ 所有系统的存储数据需求必须说明为属性。

在识别属性的过程中，为避免找出冗余的或不正确的属性，应注意以下问题：
§ 对于应用论域中的某个实体，如果不仅其取值有意义，而且它本身也有必要独立存在，那么，应将该实体作为一个对象，而不宜作为另一个对象的属性。
§ 对象的导出属性应当略去。例如，“年龄”是由属性“出生年月”与系统当前日期导出。因此，“年龄”不应作为人的基本属性。
§ 在分析阶段，如果某属性描述对象的外部不可见状态，应将该属性从分析模型中删去。
如果在标识属性的过程中发生以下情况，应考虑调整对象识别的结果：
§ 如果属性只适应于对象的某些实例，而不适应于对象的另外一些实例，则往往意味着存在另一类对象，而且这两类对象之间可能存在着继承关系。
§ 仅有一个属性的对象可以标识为其它对象的属性。
§ 对于对象的某一个属性，如果该对象的某一个特定实例针对该属性有多重属性值，则应当将该对象分为几个对象。
通常，属性放在哪一个类中应是很明显的。较一般的属性应放在一般化－特殊化结构中较高层的类或对象中，较特殊的属性应放在较低层的类或对象中。

数据视图 ERD 中实体可能对应于某一对象。这样，实体属性就会简单地成为对象属性 。如果实体（如人）不只对应于一类对象，那么这个实体的属性必须分配到 OOA 模型的不同类的对象之中。

② 找出实例连接

实例连接是一种表示应用论域的映象模型，它表明某一对象为完成其职责，需要其它对象的参与。这一步骤要为每一个实例连接定义重复度(1:1, 1:m, m:m)。当标识完实例连接之后，可能需要对对象进行调整。
§ 查看是否有什么属性描述的是多对多的实例连接，如果有，则可能需要增加一个新的“交互”对象。

§ 查看在同一个类各个实例之间的实例连接。对于每一个连接，看是否有属性描述它。如果有，应当建立新的类或对象。例如，“婚姻”是人与人之间的关系，为描述婚姻发生的地点、时间等，除了“人”这个对象外，还需要建立“婚姻”事件对象。

§ 检查对象之间的实例连接情况。如果两个对象之间有两个或两个以上的实例连接时，可能需要加入一个类或对象来区分这些实例连接的含义。

(5) 标识服务
下一个层次称为服务层。在这一层标识对象所执行的服务以及对象之间传递的消息，建立对象之间的动态关系,其目的在于定义对象的行为和对象之间的通信(消息连接),说明所标识的各种对象是如何共同协作，使系统运作起来。定义服务的步骤如下：
① 标识在每个对象中必须封装的一组服务；
§ 简单的服务：每一个类或对象都应具备的服务，这些服务包括：建立和初始化一个新对象，建立或切断对象之间的关联，存取对象的属性值，释放或删除一个对象。

§ 复杂的服务：分为两种。计算服务：利用对象的属性值计算，以实现某种功能；监控服务：处理对外部系统的输入／输出，外部设备的控制和数据的存取。

② 将服务与对象的属性相比较，验证其一致性。如果已经标识了对象的属性，那么每个属性必然关联到某个服务,否则该属性就形同虚设,永远不可能被访问;
③ 画出对象之间的消息通信路径，协调系统的行为。
通常消息有以下几类：发送对象激活接收对象；发送对象传送信息给接收对象；发送对象询问接收对象；发送对象请求接收对象提供服务。这几种类型可根据描述对象之间动作关系的动词和句型来区分。对象之间的通信只能通过消息的发送和接收来完成。消息由发送对象传给接收对象，其中包含有发送者希望完成的服务名和相关的参数。
§ 自底向上的方法：访问每一个对象，给出在对象生存期中从建立到消亡的所有状态。每一状态的改变都关联到对象之间消息的传递。从对象着手，逐渐向上分析。
§ 自顶向下的方法：一个对象必须识别某个系统中发生或出现的事件，产生发送给其它对象的消息，由那些对象作出响应。所以对象应能够询问需要执行什么服务，以便接收、处理、产生每个消息。它是从系统行为着手，然后逐渐分析到对象。
当一个对象将一个消息传送给另一个对象时,另一个对象又可传送一个消息给另一个对象，如此下去就可得到一条执行线索。检查所有的执行线索，确定哪些是关键执行线索(Critical Threads of Execution)。这样有助于我们检查模型的完备性。

应当指出的是,从服务层中的消息连接可以看到一个对象在什么状态下对哪个消息做出怎样的反应。也就是说，每个对象被看成一个自动机。系统中所有这样的自动机构成了描述系统动态行为的基础。然而，这种零散的说明让人很难从中形成一个总体的概念。因此，Coad 与 Yourdon 加上了事件－响应对象交互图 EROI (类似于 OMT 的事件追踪图)，用来标识和描述对象之间的相互通信。对于每一个事件，EROI 图表明了由哪一个对象来识别事件的发生，产生什么消息；其它哪些对象接收这些消息，并产生什么响应。EROI 图的表示方法如图 6.26 所示。

图 6.26 所示。

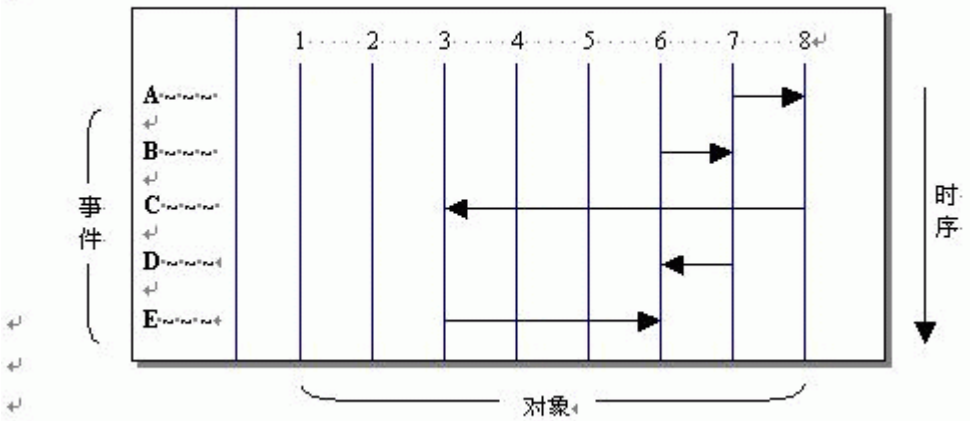


图 6.26 事件－响应对象交互图 EROI

(6) 标识主题

对于复杂的应用，可能会识别出大量的对象，但分析员同时能处理的对象却不宜过多。因此引入主题机制。

通过建立多个主题，可以处理规模比较大的复杂模型，降低系统的复杂性。每个主题可以看作为一个子模型，甚至是一个子系统。在分析的某一时刻，分析人员只需重点关注某个特定的主题域。

可以依据子论域、子系统，甚至组织或地域区分主题。对于非常庞大与复杂的系统，可以建立多级主题，在主题中可以包含另一主题，形成一个层次结构。但要避免将多级主题变成层次分解。

不同的主题之间可以有重叠部分，用自顶向下（先建立主题层）或自底向上（先建立对象－类层和结构层)方式都可以建立主题。何时将主题加入到分析模型中，应当根据待开发系统的开发计划而定。如果开发计划中规定的类或对象的个数不超过 40，可以在完全了解所有的类或对象之后再将主题加入。如果系统开发计划规定的系统规模较大，主题则应尽快加入，以便将问题论域划分成若干问题子论域，进而建立工作单元。为作到这一点，需要由有经验的分析员很快标识出类和结构，粗略确定一些暂时的主题，将它们分发给负责开发的各个小组去继续进行分析。在深入了解问题论域和应用系统的职责之后，还需要对那些暂时的主题进行修正及调整。

7. Coad 与 Yourdon 面向对象设计方法

OOD 模型类似于构造蓝图，以最完整的形式全面地定义了如何用特定的实现技术建立起一个目标系统。在 OOA 模型和 OOD 模型中使用了共同的表示法。这有助于从分析到设计的转换，并有助于在当前的设计和实现中维护 OOA 模型。

与 OOA 模型一样，OOD 模型也有 5 层结构，又被划分成了 4 个组成部分：问题论域、用户界面、任务管理和数据管理。这些组成部分把实现技术隐藏起来，使之与系统的基本问题论域行为分离开来。这种策略能够帮助提高产品的可复用性，有助于产品的升级换代。

在 OOA 中，我们实际上只涉及到问题论域部分，其它三个部分是在 OOD 中加进来的。由于问题论域部分包括与我们所面对的应用问题直接有关的所有类和对象。由于识别和定义这些类和对象的工作在 OOA 中已经开始，这里只是对它们做进一步的细化。

在其它的三个部分中，将识别和定义新的类和对象。这些类和对象形成问题论域部分与用户、与外部系统和专用设备，以及与磁盘文件和数据库管理系统的界面。Coad 与 Yourdon 强调这三部分的作用主要是保证系统基本功能的相对独立，以加强软件的可复用性。假如外部的通信系统更新了，相应的通信协议也应有所变化。在这种情况下，我们只需修改任务管理部分中的某些类和对象，而不必对其它几个部分做任何修改。

(1) 问题论域部分(PDC)的设计

完整的未经改动的 OOA 模型将成为初始的 OOD 模型的 PDC 部分。然后根据实现技术及实现方面的限制，对初始 PDC 部分的模型中的某些类与对象、结构、属性、操作进行组合与分解。但保留在 OOA 模型中所捕获到的基本的系统行为。如果使用可复用的类，那么它也要引入到 PDC 中。另外根据 OOD 的附加原则，增加必要的类、属性和关系。

① 复用设计
根据问题解决的需要，把从类库或其它来源得到的既存类增加到问题解决方案中去。既存类可以是用面向对象程序语言编写出来的，也可以是用其它语言编写出来的可用程序。要求标明既存类中不需要的属性和操作，把无用的部分维持到最小限度。并且增加从既存类到应用类之间的一般化－特殊化的关系。进一步地，把应用中因继承既存类而成为多余的属性和操作标出。还要修改应用类的结构和连接，必要时把它们变成可复用的既存类。
② 把应用论域相关的类关联起来

在设计时，从类库中引进一个根类，做为包容类，把所有与应用论域有关的类关联到一起，建立类的层次。把同一应用论域的一些类集合起来，存于类库中。

③ 加入一般化类以建立类间协议
有时，某些特殊类要求一组类似的服务。在这种情况下，应加入一个一般化的类，定义为所有这些特殊类共用的一组服务名，这些服务都是虚函数。在特殊类中定义其实现。
④ 调整继承支持级别
在 OOA 阶段建立的对象模型中可能包括有多继承关系，但实现时使用的程序设计语言可能只有单继承，甚至没有继承机制，这样就需变更 PDC 中类的层次结构。图 6.27(a) 是多继承模式。

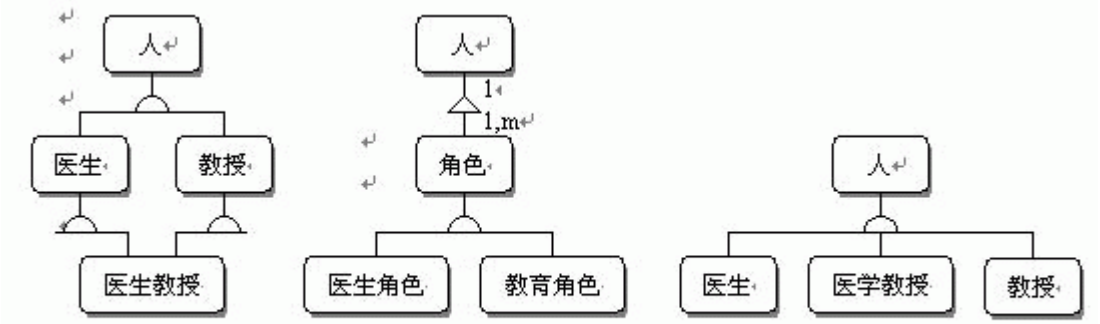


图 6.27 多继承改为单继承

§ 针对单继承语言的调整 ：利用单继承语言，可使用两种方法把多继承结构转变为单继承结构。一是把特殊类的对象看做是一个一般类对象所扮演的角色，通过实例连接把多继承的层次结构转换为单继承的层次结构，如图 6.27(b) 所示；一是把多继承的层次结构平铺，成为单继承的层次结构，如图 6.27(c) 所示。在这种情况下，有些属性或操作在同层的特殊类中会重复出现。

§ 针对无继承语言的调整 ：当使用无继承的程序设计语言时，必须把具有继承关系的类层次结构平铺开来，成为一组类和对象。一般可利用命名惯例，把这些类或对象关联起来。

⑤ 修改设计以提高性能

提高执行效率和速度是系统设计的主要指标之一。有时，必须改变问题论域的结构以提高效率。如果类之间经常需要传送大量消息，可合并相关的类，使得通信成为对象内的通信，而不是对象间的通信；或者使用全局数据作用域，打破封装的原则，以减少消息传递引起的速度损失。增加某些属性到原来的类中，或增加低层的类，以保存暂时结果，避免每次都要重复计算造成速度损失。

为提高性能，在对 OOA 模型进行大规模的改动之前，应考虑下面一些问题：

§ 如果没有性能准则，不要去人为地建立。当软件运行在一个 CPU 速度很快的计算机上，并是单机的人机交互时，大多数的时钟周期都用在了等待用户输入上。

§ 不要认为象 C++之类的 OOPL 就一定效率不高。有一些事实表明，非 OOPL 的紧凑代码的效率比 OOPL 的效率高近 10 倍，但在大多数情况下，OOPL 的效率损失约为 10%。而且用非 OOPL 编程会令程序员非常疲劳，容易出错。

§ 提高一个现存系统的工作效率比重新设计一个高效的系统要容易。一开始应当建立一个原始的简单的设计，实现和调试不会太困难。如果对设计有性能要求，只需加入少量的工作就可以了。

§ 通常系统 80%的开销都集中在 20%的代码段上。与其为了尽量处处节省系统开销而破坏完善的系统结构，还不如找出系统开销最集中的地方，只对那部分做优化。

§ 预测软件开销集中在什么地方是困难的，进行优化最有效的方法是在系统运行时使用性能监测工具对系统进行观测。一些像继承、动态绑定、消息传递等处理虽然看起来简单，但需要大量的系统开销。有的代码看起来复杂，但效率不见得低。在代码复杂性与运行的低效之间没有相关性。

§ 提高性能最好的方法是采用最出色的解决方案，而不是拼命地去节省几个微秒、几个字节。这个结论在面向对象技术出现是这样，在面向对象论域仍然是这

样。
⑥ 加入较低层的构件
在做面向对象分析时，分析员往往专注于较高层的类和对象，避免考虑太多较低层的实现细节。但在做面向对象设计时，设计师在找出高层的类和对象时，必须考虑到底需要用到哪些较低层的类和对象。
(2) 用户界面部分的设计

通常在 OOA 阶段给出了所需的属性和操作，在设计阶段必须根据需求把交互的细节加入到用户界面的设计中，包括有效的人机交互所必需的实际显示和输入。如 Windows、Pane、Selector 等。用户界面部分设计主要由以下几个方面组成。

① 用户分类

进行用户分类的目的是明确使用对象，针对不同的使用对象设计不同的用户界面，以适合不同用户的需要。分类的原则有：

§ 按技能层次分类：外行 / 初学者 / 熟练者 / 专家。

§ 按组织层次分类：行政人员 / 管理人员 / 专业技术人员 / 其它办事员。

§ 按职能分类：顾客 / 职员

② 描述人及其任务的场景

对以上定义的每一类用户，列出对以下问题做出的考虑：什么人、目的、特点、成功的关键因素、熟练程度以及任务场景。

③ 设计命令层

§ 研究现行的人机交互活动的内容和准则。这些准则可以是非正式的，如“输入时眼睛不易疲劳”，也可以是正式规定的；

§ 建立一个初始的命令层：可以有多种形式，如一系列 Menu Screens、或一个 Menu Bar、或一系列 Icons。
§ 细化命令层：这时，要考虑以下几个问题。

• 排列命令层次。把使用最频繁的操作放在前面；按照用户工作步骤排列。

• 通过逐步分解，找到整体-部分模式，帮助在命令层中对操作进行分块。

• 对菜单宽度与深度进行比较，把深度尽量限制在三层之内。

• 减少操作步骤：在完成必须任务的前提下，把点取、拖动和键盘操作减到最少。

④ 设计详细的交互

用户界面设计有若干原则，其中包括：

§ 一致性：采用一致的术语、一致的步骤和一致的活动。

§ 操作步骤少：减少敲键和鼠标点取的次数，减少完成某件事所需的下拉菜单的距离。

§ 不要“哑播放”：即每当用户等待系统完成一个活动时，要给出一些反馈信息，说明工作正在进展，以及进展的程度。

§ Undo：在操作出现错误时，恢复或部分恢复原来的状态。

§ 减少人脑的记忆负担：不应在一个窗口使用在另一个窗口中记忆或写下的信息；需要人按特定次序记忆的东西应当组织得容易记忆。

§ 学习的时间和效果：提供联机的帮助信息。

§ 趣味性：在外观和感受上，尽量采取图形界面，符合人类习惯，有一定吸引力。

⑤ 继续做原型

用户界面原型是用户界面设计的重要工作。人需要对提交的人机交互活动进行体验、实地操作，并精炼成一致的模式。使用快速原型工具或应用构造器，对各种命令方式，如菜单、弹出、填充以及快捷命令，做出一些可供选择的原型，让用户使用，收集用户的反映，通过修改、演示的迭代，使界面越来越有效。

⑥ 设计 HIC(人机交互)类

设计 HIC 类，首先从组织窗口和构件的用户界面界面的设计开始。窗口需要进一步细化，通常包括类窗口、条件窗口、检查窗口、文档窗口、画图窗口、过滤器窗口、模型控制窗口、运行策略窗口、模板窗口等。

每个类包括窗口的菜单条、下拉菜单、弹出菜单的定义。还要定义用于创建菜单、加亮选择项、引用相应的响应的操作。每个类还负责窗口的实际显示。所有有关物理对话的处理都封装在类的内部。必要时，还要增加在窗口中画图形图符的类、在窗口中选择项目的类、字体控制类、支持剪切和粘贴的类等。与机器有关的操作实现应隐蔽在这些类中。

⑦ 根据图形用户界面进行设计

图形用户界面区分为字型、坐标系统和事件。图形用户界面的字型是字体、字号、样式和颜色的组合。坐标系统主要因素有原点(基准点)、显示分辨率、显示维数等。事件则是图形用户界面程序的核心，操作将对事件做出响应，事件的工作方式有两种：直接方式和排队方式。所谓直接方式，是指每个窗口中的项目有它自己的事件处理程序，一旦事件发生，则系统自动执行相应的事件处理程序。所谓

排队方式，是指当事件发生时系统把它排到队列中，每个事件可用一些子程序信息来激发。

(3) 任务管理部分的设计

所谓任务，是进程的别称，是执行一系列活动的一段程序。当系统中有许多并发行为时，需要依照各个行为的协调和通信关系，划分各种任务，以简化并发行为的设计和编码。而任务管理主要包括任务的选择和调整，它的工作有以下几种。

① 识别事件驱动任务

一些负责与硬件设备通信的任务是事件驱动的，也就是说，这种任务可由事件来激发，而事件常常是当数据到来时发出一个信号。

② 识别时钟驱动任务

以固定的时间间隔激发这种事件，以执行某些处理。某些人机界面、子系统、任务、处理机或与其它系统需要周期性的通信。

③ 识别优先任务和关键任务 根据处理的优先级别来安排各个任务。在系统中，有些操作具有高优先级，因此必须在很强的时间限制内完成；有些操作具有较低的优先级，可进行时间要求较低的处理(如后台处理)。通常需要有一个附加的任务，把各个任务分离开来。

所谓关键任务是对系统的成败起关键作用的处理。必须使用附加的任务来分离这种任务，并对其安全性仔细进行设计、编程和测试。

④ 识别协调者

当有三个或更多的任务时，应当增加一个附加任务，起协调者的作用。它的行为可以用状态转换矩阵来描述。这种任务仅用于协调任务。

⑤ 评审各个任务

必须对各个任务进行评审，确保它能满足选择任务的工程标准——事件驱动、时钟驱动、优先级 / 关键任务或协调者。

⑥ 定义各个任务

定义任务的工作主要包括：它是什么任务、如何协调工作及如何通信。

§ 它是什么任务：为任务命名，并简要说明这个任务。

§ 如何协调工作：定义各个任务如何协调工作。指出它是事件驱动还是时钟驱动。对于事件驱动的任务，描述激发该任务的事件；对于时钟驱动的任务，指明激发之前所经过的时间间隔，同时指出是一次性的还是重复性的时间间隔。

§ 如何通信：定义各个任务之间如何通信。任务从哪里取值，结果送往何方。

§ 一个模版：任务的定义—任务名、描述、优先级、包含的操作、经由谁通信。

(4) 数据管理部分的设计

数据管理部分提供了在数据管理系统中存储和检索对象的基本结构，包括对永久性数据的访问和管理。它分离了数据管理机构所关心的事项，包括文件、关系型DBMS 或面向对象 DBMS 等。

数据管理方法主要有 3 种：文件管理、关系数据库管理和面向对象库数据管理。

§ 文件管理：提供基本的文件处理能力。

§ 关系数据库管理系统(RDBMS)：关系数据库管理系统建立在关系理论的基础上，它使用若干表格来管理数据。通常根据规范化的要求，可对表格和它们的各栏重新组织，以减少数据冗余，保证修改一致性数据不致出错。规范化的要求用“范式”来定义。

§ 面向对象数据库管理系统(OODBMS)：通常，面向对象的数据库管理系统以两种方法实现：一是扩充的 RDBMS，二是扩充的面向对象程序设计语言(OOPL)。

扩充的 RDBMS 主要对 RDBMS 扩充了抽象数据类型和继承性，再加上一些一般用途的操作来创建和操纵类与对象。扩充的 OOPL 对面向对象程序设计语言嵌入了在数据库中长期管理存储对象的语法和功能。这样，可以统一管理程序中的数据结构和存储的数据结构，为用户提供了一个统一视图，无需在它们之间做数据转换。

(5) 程序设计语言的影响

详细的面向对象设计是与语言有关的。这样，可以有效地为过程型语言、面向软件包的语言、面向对象的语言所用。

一般地，所有的语言都可以完成面向对象实现，但某些语言能够提供更丰富的语法，能够显式地描绘在面向对象分析和面向对象设计过程中所使用的表示法。

① 面向对象设计与过程型语言

过程型语言，如 C、Pascal、FORTRAN、COBOL，只直接支持过程抽象，但可以增加数据抽象及封装(如利用结构化设计的信息隐蔽模块)。虽然某些公共部分可以作为单独的子程序，但无法明确地表示继承性。也无法明确支持整体与部分、类与成员、对象与属性等关系。也许，面向过程型语言的面向对象设计尽管在技术上不令人满意，但它确实还是能成为一种实用的且可行的方法：从面向对象分析，到面向对象设计，再到具有面向对象特性的过程型语言。

② 面向对象设计与基于对象的语言

基于对象的语言，也叫做面向软件包的语言，如 Ada 等，能够直接支持过程抽象、数据抽象、封装和对象与属性关系。虽然某些公共部分可作为单独的子程序，但它仍无法明确地表示继承性，也无法明确地表示类与成员、整体与部分的关系。也许，基于对象语言的面向对象设计比较符合人的习惯，它代表着一种可行的开发方法： 从面向对象分析，到面向对象设计，再到具有面向对象特性的基于对象的语言。

③ 面向对象设计与面向对象的程序设计语言

面向对象的程序设计语言，包括 C++、Smalltalk、Actor、Objective-C、Eiffel、Java 等，都直接支持过程抽象、数据抽象、封装、继承、以及对象与属性、类与成员关系。虽然它们不明确地支持整体与部分关系，但可以方便地表示组装对象。因此，从面向对象分析，到面向对象设计，再到面向对象程序设计语言(OOPL)是一种与表示法十分一致的策略。

④ 面向对象设计与面向对象数据库语言(OO-DBL)

面向对象数据库管理系统(OO-DBMS)及其语言(OO-DBL)，是面向对象程序设计语言(OOPL)与数据管理能力的组合。OO-DBMS 有四种不同的体系结构：

§ 大属性：扩充关系型 DBMS，使容纳大属性，如一个文档。

§ 松散耦合：一个 OOPL 与大量的 DBMS 组合在一起。

§ 紧密耦合：一个 OOPL 与某个专用的 DBMS 集成为一个系统。

§ 扩充关系型：扩充关系型 DBMS，使容纳“过程”之类的属性。

紧密耦合体系结构在程序设计和数据操纵中使用了同一种语言，它更能显式地表达面向对象分析和面向对象设计的语义。

8. Booch 的方法

就面向对象的方法而言，Booch 是最早的倡导者之一，他的工作可回溯到 80 年代初期。

(1) Booch 方法的设计过程

Booch 认为软件开发是一个螺旋上升的过程。在这个螺旋上升的每个周期中，有以下几个步骤：

① 发现类和对象

从说明应用问题的词法和概念中识别对象，通过对具体对象的抽象化来发现类。

② 确定它们的含义

进行行为分析，描画相关的场景，考虑已识别的类和对象在完成系统功能上应承担的责任和所起到的作用，在这个基础上确定每一类的属性和操作。

③ 找出它们之间的相互关系

找出类与类，对象与对象之间的相互关系。密切相关的一些对象协同作业，以完成部分的系统功能，同时也构成系统的一个必要组成部分，Booch 称之为“机构”。识别这样的机构也是这一步的一个重要目的。

④ 说明每一个类和对象的界面和实现

说明每一类的界面和实现，同时将类和对象分配到不同的模块中，将可同时执行的进程分配到不同的处理机上。这一步是对已有定义的细化和完善过程，往往有助于发现新的类和对象，因而导致下一周期的开发工作。

Booch 对每一步的目的，具体做法，最后的产品，以及检验方法都做了详细的讨论。图 6.28 给出了 Booch 的面向对象的开发模型。这个模型分为逻辑设计和物理设计两个部分。

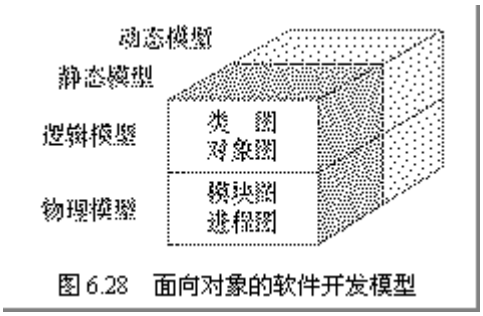


图 6.28 面向对象的软件开发模型

逻辑设计部分包括两个文件：类图和对象图，着重于类和对象的定义。物理设计部分也包括两个文件：模块图和进程图，针对着软件系统的结构。

Booch 还区分静态模型和动态模型。静态模型侧重于系统的构成和结构，而动态模型则侧重于系统在执行过程中的行为。除了上面提到的几个基本文件以外，Booch 的方法还包括状态迁移图和时序图，这两个文件主要用于描述系统的动态行为。

(2) Booch 方法的基本的模型

在 Booch 方法中，用于说明系统要求的表示方法和手段非常丰富，相当灵活。

① 类图

类图用于表示类的存在以及类与类之间的相互关系，是从系统构成的角度来描述正在开发的系统。图 6.29 所给出的以虚线为边界的云状图符（云图）表示一个类。它的名字，属性和操作则可列于其中。所有的图形表示都伴随有详细的文字说明，不必将全部细节都画在图中。



图 6.29 类的图形表示

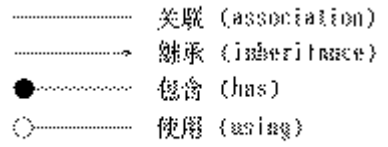


图 6.30 类与类之间的相互关系图示

一个类的存在不是孤立的。类与类之间以不同方式互相合作，共同完成某些系统功能。图 6.30 给出了几种基本的关系及其图形表示。关联关系表示两个类之间存在着某种语义上的联系，其真正含义要由附加在横线之上的一个短语来予以说明。在表示继承关系的图符中，箭头由子类指向基类。在表示包含关系的图符中，带有实心圆的一端表示整体，相反的一端表示部分。在表示使用关系的图符中，带有空心圆的一端连接在请求服务的类，相反的一端连接在提供服务的类。图 6.31 给出了一个有关温室管理系统的类图。

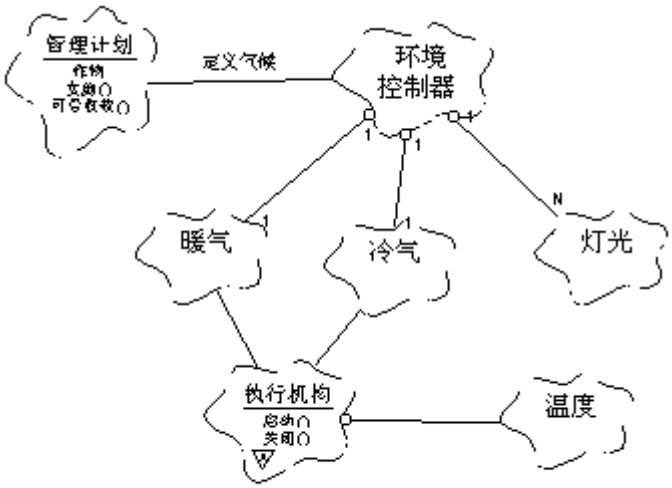


图 6.31 温室管理系统的类图

② 对象图

对象图用于表示对象的存在以及它们间的相互关系。在系统的生存期中，类的存在基本上是稳定的，而对象则不断地从产生到消灭，经历着一系列的变化。一个对象图则是描述在这个过程中某一时刻的场景，也可以说是用来说明决定系统行为的基本结构。

从图 6.32 中的对象图中可以看到，用实线的云状图符表示对象，右下侧有黑影的云状图符表示系统程序库中的公用程序，云状图符之间的连线及其之上的箭头

表示消息的传送和 方向，为表示消息传送的次序，序列号码附加在每个消息之间。

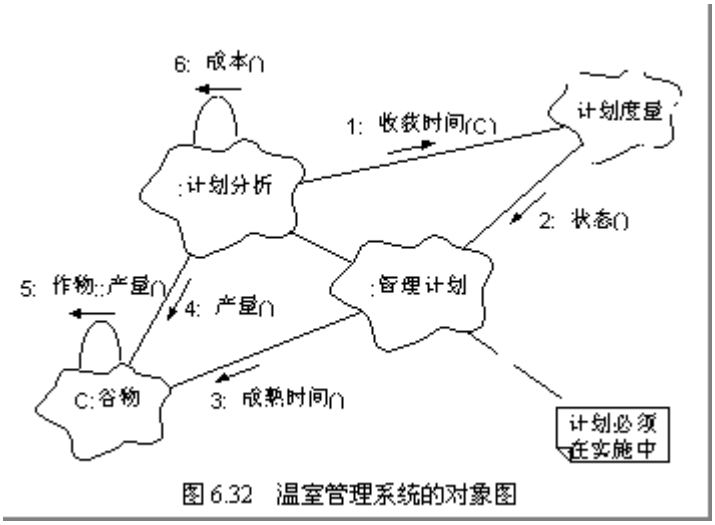


图 6.32 温室管理系统的对象图

(3) 状态迁移图

状态迁移图用来说明每一类的状态空间、触发状态迁移的事件，以及在状态迁移所执行的操作。它提供了描述一个类的动态行为的手段。

图 6.33 给出了“环境控制器”类的状态迁移图。从图中可以看到 Booch 的表示方法非常类似于 OMT 的表示方法。同样地，Booch 也主张采用类似结构化的方法来减少状态迁移图的复杂性。

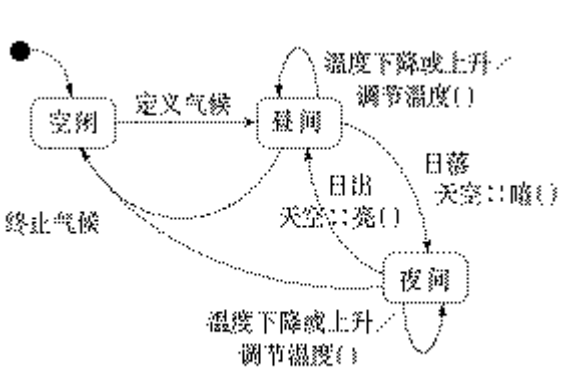


图 6.33 “环境控制器”的状态迁移图

(4) 交互作用图

Booch 方法中的交互作用图不仅在概念上，而且在表示方法上都与 OMT 的事件追踪图十分相似。不同的是 Booch 的交互作用图主要表示操作而不是事件。交互作用图用于追踪系统执行过程中的一个可能的场景，也就是几个对象在共同完成某一系统功能中所表现出来的交互关系。Booch 指出交互作用图实际上是另一种形式的对象图，完全有可能利用软件工具在一个图的基础上自动地生成另一个。图

6.34 的交互作用图与图 6.32 的对象图表现系统执行过程中的同一场景，但以不同的形式来表达。交互作用图可以清楚地展示消息传送的序列。而对象图则能够表现较为复杂的操作调用。因而，Booch 的方法包括这两种图。

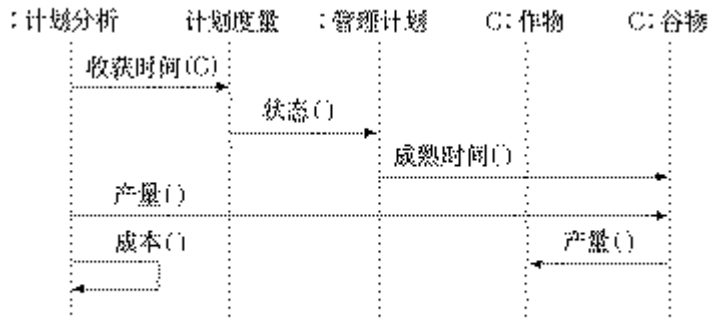


图 6.34 温室管理系统的交互作用图

(5) 模块图

模块图在系统的物理设计中说明如何将类和对象分配到不同的软件模块中。具体方法与最后代码编写时所采用的程序设计语言有关。在多数语言中，文件就是基本的模块。

图 6.35 给出了一个模块图的例子。图中的方形图符用以表示声明文件，其右下策的黑影表示相应的定义文件。从一个文件图符到另一个文件图符的箭头表示两者之间的编译依赖关系，也就是说，在箭头所指向的文件编译完成之后，相反一端的文件不可进行编译。

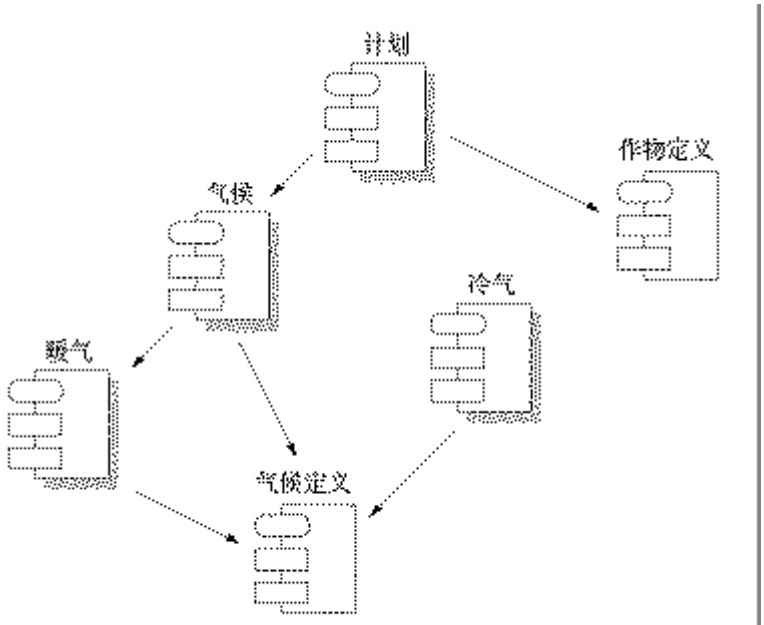


图 6.35 温室管理系统的模块图

除了文件层的模块图以外，Booch 的方法还建议用子系统层的模块图来描述系统的主要部分之间以及与外部系统(如数据库)之间的关系

(6) 进程图

进程图在系统的物理设计中说明如何将可同时执行的进程分配到不同的处理机上。即使对于运行于单处理机之上的系统，进程图也是有用的，因为它可以表示同时处于活动状态的对象，以便决定进程调度方法。图 6.36 给出的是温室管理系统的进程图。立方形的图符代表进程，它们之间的连线表示进程之间的通信关系。

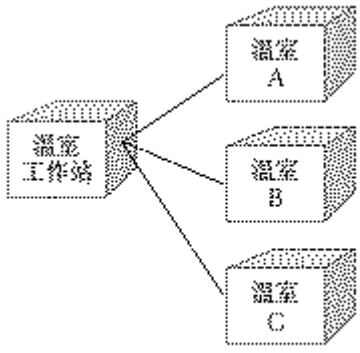


图 6.36 温室管理系统的进程图

总之，Booch 的方法内涵丰富，涉及到面向对象的软件系统所有各个方面，是广为使用的几个面向对象的方法之一。Booch 不仅建立了开发方法，还提出了对设计人员的技术要求，以及在开发过程的不同阶段（分析、设计、代码编写、测试与集成）中资源与人力的分配。此外，他还讨论了设计阶段的工作何时结束，实现阶段的工作何时结束这样一些较为困难的问题。

三、例题分析

【例 1】请说明下面有关范型的叙述的正确答案。

(1) 问题的解决是基于规则的，它把有关问题的知识分解成一组具体规则，用语言的 if_then 等结构来表示这些规则。

(2) 问题的解决把软件视为由一系列步骤构成的算法。每一步骤都是带有预定输入和特定输出的一个过程，连贯起来产生合理的稳定的贯通于整个程序的控制流。

(3) 把一个问题分解成独立执行的模块。让不只一个程序(进程)同时运行。这些进程互相配合，解决问题。

(4) 把标识和模型化问题论域中的主要实体做为系统开发的起点，主要考虑对象的行为而不是必须执行的一系列动作。 供选择的答案：

① 面向存取 ② 面向对象 ③ 过程性 ④ 逻辑性 ⑤ 函数型

⑥ 面向进程 ⑦ 说明型 ⑧ 原型

答案：(1) ④ (2) ③ (3) ⑥ (4) ②

分析：逻辑性的范型把有关问题的知识分解成一组具体的规则，这些规则常常是用一种语言的“if_then”结构等来表示。过程性的范型产生过程的抽象，这些抽象把软件视为处理流，定义构成一系列步骤的算法，每一步骤都是带有预定义输入和特定输出的一个过程，把这些步骤串联在一起可产生合理的稳定的贯通于整个程序的控制流。面向对象范型把标识和模型化问题论域中的主要实体做为系统开发的起点，主要考虑对象的行为而不是必须执行的一系列动作。面向对象系统中的对象是数据抽象与过程抽象的一个混合体。表示这些实体的数据抽象是面向对象设计过程的主要产品，系统的状态保存在各个数据抽象的核心所定义的数据存储中。控制流被分成块，并被包括在各个在数据抽象上的各个操作里面。不像在过程性范型里那样，把数据从一个过程传送到另一个过程，而是控制流从一个数据抽象被传送到另一个数据抽象。面向进程的范型把一个问题分解成独立执行的模块。在某种意义上来说，就是让不只一个程序同时运行。这些程序，或更正确地说，这些进程互相配合，从而解决问题。可以认为一个进程在面向进程开发模式中就是一个模块(构件)。面向进程开发模式产生的主要的块是进程。在一个进程中的活动总的来说是独立于其它进程的那些活动的，除非要求从其它进程得到信息，或为其它进程提供信息。甚至可以异步处理，仅需要进程暂停发送或接收信息。在面向对象范型中，各个对象是相对独立的，但也存在单线索控制。面向进程范型支持与面向对象范型相同的封装，但可继续提供多线索(线程)执行。

【例 2】对象是面向对象范型的（ A ）。每个对象可用它自己的一组（ B ）和它可以执行的一组(C)来表征。应用执行对象的(C)可以改变该对象的(B)。它的应用必须通过（ D ）的传递。可以认为，这种（ D ）的传递大致等价于过程性范型中的函数调用。某些语言提供了特殊功能，允许对象引用自己。若一个对象没有显式地被引用，则可让该对象（ E ）。

供选择的答案：

A. ① 基本单位 ② 最小单位 ③ 最大单位 ④ 语法单位

B ~ C. ① 行为 ② 功能 ③ 操作 ④ 数据 ⑤ 属性

D. ① 接口 ② 消息 ③ 信息 ④ 操作 ⑤ 过程

E. ① 撤消 ② 歇着 ③ 缺省 ④ 隐式引用 ⑤ 引用自己

答案：A. ①, B. ⑤, C. ③, D. ② E. ③

分析：对象是面向对象范型的基本单位。每个对象可用它自己的一组属性和它可以执行的一组操作来表征。应用执行对象的操作可以改变该对象的属性，属性一般只能通过操作来改变。它的应用必须通过消息的传递。发送给一个对象的消息定义了一个方法名和一个参数表（可能是空的），并指定某一个对象。而由一个对象接收的消息则引用消息中指定的方法的代码，并将实际参数与参数表中相应的形式参数结合起来。接收对象对消息的处理可能会改变对象中的状态，即改变接收对象的属性，并发送一个消息给自己或另一个对象。可以认为，这种消息的传递大致等价于过程性范型中的函数调用。然而，执行方法得到消息结果的目的是想修改相关对象的内部状态，而不只是修改变元并返回它们。某些语言提供了特殊功能，像 smalltalk 提供了 self，允许对象引用自己。若一个对象没有显式地被引用，则可让该对象缺省。

【例 3】在面向对象软件开发过程中特别重视复用。软件构件应独立于当初开发它们的应用而存在。在以后的应用开发中，可以调整这些独立构件以适应新问题的需要。因此，应使得类成为一个（ A ）的单元。这样就有一个（ B ）生存期问题。（ B ）生存期有自己的步骤，与任一特定应用的开发（ C ）。按照这些步骤，可以完整地描述一个基本（ D ）。而不仅仅考虑当前正在开发的系统。系统开发的各个阶段都可能会标识新的类。随着各个新类的标识，（ B ）生存期引导开发工作逐个阶段循序渐进。

在设计与实现类时，应尽可能利用既存类提供为当前应用所需要的功能，利用既存类的三个可能途径是：（E）复用既存类；对既存类进行（ F ）以得到满足要求的类；重新开始进行开发。

供选择的答案：

A. ① 可复用 ② 可测试 ③ 可适用 ④ 可靠

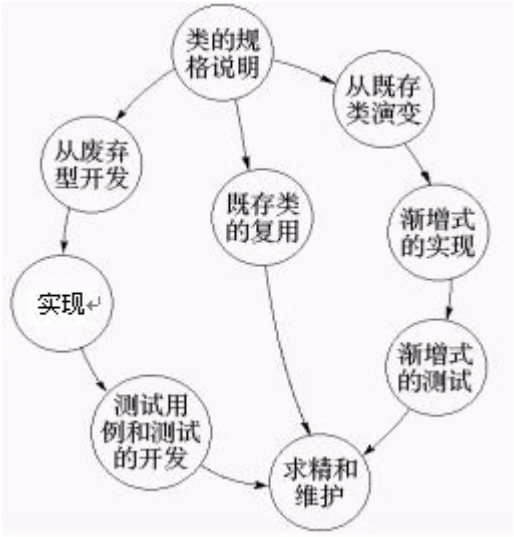
B. ① 应用 ② 寿命 ③ 类 ④ 软件

C. ① 相关 ② 密切相关 ③ 负相关 ④ 无关

D. ① 概念 ② 实体 ③ 事件 ④ 事情

E, F. ① 修改 ② 更新 ③ 照原样 ④ 演化

答案：A. ①, B. ③, C. ④, D. ②, E. ③, F. ④



分析：在面向对象软件开发过程中特别重视复用。软件构件应独立于当初开发它们的应用而存在。构件的开发瞄准某些局部的设计和实现，它们可用于当前问题的解决，但为了在以后的项目中使用，它们还应当足够通用。在以后的应用开发中，可以调整这些独立构件以适应新问题的需要。为使得类成为一个可复用的单元，有一个类生存期的问题。

类生存期与应用生存期交叉。在应用生存期的每一个阶段都可做类的标识。类生存期有自己的步骤，与任一特定应用的开发无关。按照这些步骤，可以完整地描述一个基本实体。而不仅仅考虑当前正在开发的系统。系统开发的各个阶段都可能会标识新的类。随着各个新类的标识，类生存期引导开发工作逐个阶段循序渐进。 在设计与实现类的时候，有三种利用既存类的途径：

§ 原封不动地复用既存类。

§ 对既存类进行演化以得到满足要求的类。

§ 重新开始进行开发。

【例 4】类常常被看做是一个抽象数据类型的实现，更合适的是把类看做是某种（ A ）的一个模型。事实上，类是单个的（ B ）语义单元。类的用户能够操纵的操作叫做类的（ C ）。类定义的其余部分给出数据定义和辅助功能定义，包括类的实现。

类的实现常常包括了其它类的实例，这些实例（ D ）被其它对象存取，包括同一个类的其它实例。类的实现可能还包括某些私有方法，实现它们的类可以使用，而其它任何对象都不能使用。

类，就它是一个数据值的聚合的意义上来看，与 Pascal 中的记录或 C 中的结构类似，但又有差别。类扩展了通常的记录语义，可提供各种级别的（ E ）。类不同于记录，因为它们包括了操作的定义，这些操作与类中声明的数据值有相同的地位。

供选择的答案：

A. ① 功能 ② 概念 ③ 结构 ④ 数据

B. ① 语法 ② 词法 ③ 语义 ④ 上下文环境

C. ① 界面 ② 操作 ③ 行为 ④ 活动

D. ① 可自由地 ② 可有控制地 ③ 可通过继承 ④ 应受保护不

E. ① 可移植性 ② 可重复性 ③ 可访问性 ④ 继承性

答案：A. ②, B. ③, C. ①, D. ④, E. ③

分析：类常常被看做是一个抽象数据类型的实现。这个定义不太够，更合适的是把类看做是某种概念的一个模型。事实上，类是单个的语义单元，它可以更自然地管理系统中的对象，匹配数据定义与操作。许多面向对象范型的语言都提供数据抽象机制。这个机制为类定义提供了一个手段，以指明类的用户能够操纵的操作。这组操作叫做类的界面。类定义的其余部分给出数据定义和辅助功能定义，包括类的实现。这种分离把类的用户与类内部修改的影响隔离开来。类的实现常常包括了其它类的实例，它们往往提供了新类所需要的服务。这些实例应受保护不被其它对象存取，包括同一个类的其它实例。类的实现可能还包括某些私有方法，实现它们的类可以使用，而其它任何对象都不能使用。

类，就它是一个数据值的聚合的意义上来看，与 Pascal 中的记录或 C 中的结构类似，但又有差别。类扩展了通常的记录语义，可提供各种级别的可访问性。也就是说，记录的某些成份可能是不可访问的，而这些成份对于本记录型来说具有可访问性。类不同于记录，因为它们包括了操作的定义，这些操作与类中声明的数据值有相同的地位。

【例 5】 有一种非形式的技术，对于捕获信息有时很有用，它就是 CRC 卡片。CRC 是(A)、(B)和(C)的缩写。它可以用来组织在每一个子系统中的类。以 CRC 卡片为辅助工具的设计有以下几个步骤：识别(A)和(B)，分配(B)，找寻(C)， 细化。CRC 的作者强调模拟在执行每个基本功能时系统内部出现的(D)，以此推动细化工作的进行。在这个过程中，CRC 卡片是十分重要的一个工具。

用 CRC 卡片来进行设计，既不是传统的“自上而下”，也不是“自下而上”，而是从已知到未知的逐步(E)的过程。

供选择的答案：

A ~ C. ① 控制 ② 协作者 ③ 可靠性 ④ 类

⑤ 计算 ⑥ 职责 ⑦ 比较 ⑧ 上下文环境

D. ① 场景 ② 算法 ③ 进程 ④ 变换

E. ① 演化 ② 进展 ③ 展开 ④ 认识

答案：A. ④, B. ⑥, C. ②, D. ①, E. ③

分析：CRC 是 Class(类)、Responsibility(职责)和 Collaborators(协作者)的缩写。这个技术是由 Beck 等开发的，在标识实体和定义界面的行为时在小组对话中使用。

由于它的简单有效，以后逐渐成为职责驱动方法的一部分，并结合到其它的面向对象方法中去。CRC 卡片的特点是用人格化的方法，将软件系统中的每个部件，即类，看成一个独立的个体，在探索如何与其它个体携手合作完成某一系统功能中逐步完成自身的定位。为了便于分组，查找，以及修改。每张卡片分为三个部分，分别记录一个类的名称，它的主要职责和协作者。以 CRC 卡片为辅助工具的设计有以下几个步骤：

(1) 识别类和职责：CRC 卡片的作者建议使用自然语言的分析方法(LIA)，从用户需求规格说明书中的名词，物理和概念的实体中发现有用的类。引进适当的基类或子类，以形成继承关系的层次结构。然后，从用户的需求说明书中寻找对有关的信息和行为的描述，以发现职责。职责就是需做的事，必须完成的任务，后要解决的问题，但不涉及如何实现。

(2) 分配职责：将职责分配到类，并记录在相应的卡片上。CRC 的作者指出应尽可能确保行为与有关信息不要分开,与同一实体的信息要集中在一起。如有必要，有些职责可以由几个类共同承担。

(3) 找寻协作者：任何一个类在完成自己的职责时往往都需要其它类的协作。这一步就是要找到与每一个类协作的伙伴，并记录在相应的卡片上。具体做法是依次检查每一类所承担的每一项责任,看是否需要其它类的帮助来完成。如果需要，进一步确定需要其做什么。

(4) 细化：CRC 的作者强调模拟在执行每个基本功能时系统内部出现的场景，以此推动细化工作的进行。

通常在模拟一个场景的过程中，每当一个类开始“执行”时，它的卡片就被拿出来加以讨论，当“控制”传送到另一个类时，我们的注意力就从前一张卡片转移到另一张上去了。不同的场景，包括例外和出错状况，都应逐一加以模拟。在这个过程中，可以验证已有的定义，不断发现新的类、职责以及伙伴。

用 CRC 卡片来进行设计，既不是传统的“自上而下”，也不是“自下而上”，而是从已知到未知的逐步展开的过程。场景模拟有助于以试验性的方式来从事设计。由于检索卡片灵活、易于修改，不同的设计方案可以很快地形成，并进行检验。

【例 6】论域分析的(A)和对应用分析和高层设计的(B)就构成问题论域的模型。已有许多建立这种模型的技术，一种特别适用的技术就是语义数据模型。

语义数据模型来源于 Codd 的(C)数据模型和实体—联系模型，并对这类模型进行了扩充和一般化。语义数据模型可以表达问题论域的内涵，还可以表示复杂对象和对象之间的联系。语义数据模型与(C)数据模型本来都是在(D)设计时使用的，但它们的范围已经扩展到系统的开发。作为(D)结构标准的 ANSI / SPARC 建议提出了三层模型：外部模型、概念模型和(E)模型。这三层可以被映象到面向对象设计的三个层次上去。外部模型与概念模型层相当于高层设计阶段。

供选择的答案：

A, B. ① 控制 ② 输出 ③ 输入 ④ 处理

⑤ 计算 ⑥ 解释 ⑦ 比较 ⑧ 创建

C. ① 变换 ② 关系 ③ 抽象 ④ 事务

D. ① 网络 ② 程序 ③ 算法 ④ 数据库

E. ① 低层 ② 底层 ③ 内部 ④ 存储

答案：A. ③, B. ②, C. ②, D. ④, E. ③

分析：论域分析的输出和对应用分析和高层设计的输入就构成问题论域的模型。一种特别适用的技术就是语义数据模型。语义数据模型来源于 Codd 的关系数据模型和实体—联系模型，并对这类模型进行了扩充和一般化。语义数据模型可以表达问题论域的内涵，还可以表示复杂对象和对象之间的联系。语义数据模型与关系数据模型本来都是在数据库设计时使用的，但它们的范围已经扩展到系统的开发。作为数据库结构标准的 ANSI / SPARC 建议提出了三层模型：外部模型、概念模型和内部模型。这三层可以被映象到面向对象设计的三个层次上去，如表所示。外部模型与概念模型层相当于高层设计阶段。

语义数据模型	主 要 特 征	面向对象设计
外部模型	数据的用户视图	类的定义（规格说明）
概念模型	实体及其之间联系的内涵	类之间的应用联系
内部模型	数据的物理模型	类的实现

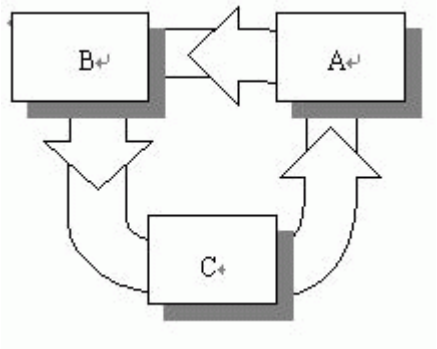
(1) 外部模型层。外部模型是来自应用的外部现实世界的视图，它反映了用户对问题的理解而不是实现者对问题的理解。在这一层开发的类应具有对应于用户活动的操作规格说明。

(2) 概念模型层。概念模型层考虑在外部模型层所标识的实体之间的联系。这些联系就是可直接观察到的交互关系。在这一层，通常可以由系统的用户来识别和

理解这些联系。 联系的重要属性是实例连接。一个联系的实例连接是指在该联系中一个实体的实例对应于该联系中其它实体的实例的数目。

(3) 内部模型层。这一层考虑实体的物理模型，这是我们生存期中的类设计阶段。物理模型包括两类属性：数据和方法。方法属性对实体的行为模型化，而数据属性对实体的状态模型化。在模型中方法分为两种：一种可作为共有界面来使用，而另一种是私有的。

【例 7】从分析到设计的过程流如图所示。一旦已经开发完成一个合理完整的(A)模型后，就要着手(B)的设计。这需要描述(B)的特征，以准确表达待实现的用户需求，以及实现需求所必须的支持环境。一旦定义了各个(B)，就开始(C)设计，这时，可利用 CRC 卡片，将属性转换为(D)，将关系转换为(E)。



供选择的答案：

A ~ C. ① 分析 ② 系统设计 ③ 模块设计 ④ 子系统设计

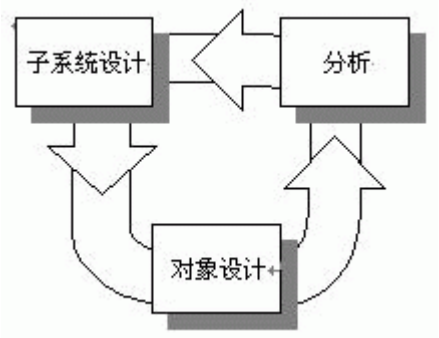
⑤ 对象设计 ⑥ 数据设计 ⑦ 操作设计 ⑧ 行为设计

D ~ E. ① 对象 ② 数据结构 ③ 算法 ④ 消息传递

⑤ 控制 ⑥ 并发处理 ⑦ 进程 ⑧ 过程

答案：A. ①, B. ④, C. ⑤, D. ②, E. ④

分析：



从分析到设计的过程流如图所示。一旦已经开发完成一个合理完整的分析模型后，就要着手子系统的设计。这需要描述子系统的特征，以准确表达待实现的用户需求，以及实现需求所必须的支持环境。

在定义子系统时，需要在用户需求的整个上下文环境中互相协调：各个用户需求分配给哪个子系统？在 OOA 中定义的对象驻留在哪个子系统内？哪些子系统并发运行？负责协调和控制它们的系统构件是谁？全局资源在子系统中如何管理？等。 在设计子系统的过程中，需要定义 4 种重要的设计构件：

§ 问题论域：负责实现用户需求的子系统。

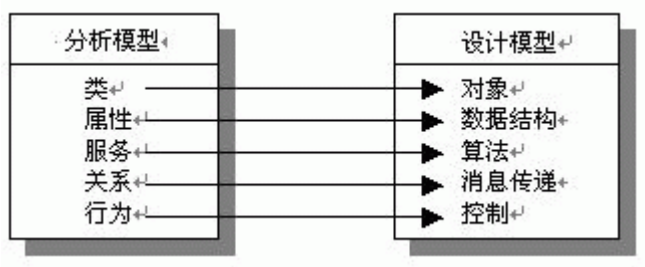
§ 人机交互：实现用户界面的子系统（包括可复用的 GUI 子系统）。

§ 任务管理：负责控制和协调各种事件驱动、时钟驱动、并发执行任务的子系统。

§ 数据管理：负责对象存储和检索的子系统。

每一个构件都可以用一些类、必须的关系和行为来定义。构件之间的关系可以通过建立每个构件的消息模型来建立。

一旦定义了各个子系统，包括上述的各个设计构件，就开始对象(类)设计，这时，可利用 CRC 卡片，将类转换为设计实现，转换工作如图所示。



四、习题

【6-1】什么叫面向对象？面向对象方法的特点是什么？为什么要用面向对象方法开发软件？

【6-2】什么是“对象”？识别对象时将潜在对象分成 7 类，试给出这 7 类对象的名称，并举例说明。

【6-3】什么是“类”？ “类”与传统的数据类型有什么关系？有什么区别？

【6-4】基于复用的面向对象开发过程分为哪几个阶段？每一个阶段需要做哪些事情？

【6-5】按照类生存期，类的开发有哪几种方式？每一种方式需要做哪些事情？

【6-6】面向对象开发方法与面向数据流的结构化开发方法有什么不同？使用面向对象开发方法的优点在什么地方？

【6-7】面向对象的程序设计语言具有数据抽象、信息隐蔽、（A）等特征。作为运算单位的对象应具有下列特性：（B）、（C）、（D）。（E）是面向对象的语言。

供选择的答案：

A：① 对象调用 ② 对象变换 ③ 非过程性

④ 信息继承 ⑤ 并发性

B ~ D：① 对象把数据和处理数据的操作结合为一体

② 在程序运行时对象都处于活动状态

③ 对象在计算中可向其他对象发送消息

④ 接受消息的对象必须给消息发送者以回答

⑤ 对象的内部状态只根据外部送来的消息才操作

E： ① C++，SMALLTALK，objectC ② C，Ada，Modula2

③ PASCAL，C++，APL ④ Ada，objectC，C

【6-8】基于复用的面向对象的需求分析过程主要分为两个阶段：论域分析和应用分析。试讨论它们各自承担什么任务？如何衔接？

【6-9】 建立分析和设计模型的一种重要方法是UML。试问UML是一种什么样的建模方法？它如何表示一个系统？

【6-10】使用面向对象设计方法进行高层设计，以建立系统的体系结构。这样的结构有哪几种主要的组成部分？每一部分所承担的职责是什么？

【6-11】由RumBaugh等人提出的一种面向对象方法叫做对象模型化技术(OMT)，即三视点技术，它要求把分析时收集的信息建立在三个模型中。第一个模型是（A），它的作用是描述系统的静态结构，包括构成系统的对象和类，它们的属性和操作，以及它们之间的联系。第二个模型是(B)，它描述系统的控制逻辑，主要涉及系统中各个对象和类的时序及变化状况。（B）包括两种图， 即（C）和（D）。（C）描述每一类对象的行为，（D）描述发生于系统执行过程中的某一特定场景。第三个模型是（E），它着重于描述系统内部数据的传送与处理，它由多个数据流图组成。

供选择的答案：

A, B, E：① 数据模型 ② 功能模型 ③ 行为模型 ④ 信息模型

⑤ 原型 ⑥ 动态模型 ⑦ 对象模型 ⑧ 逻辑模型

⑨ 控制模型 ⑩ 仿真模型

C, D： ① 对象图 ② 概念模型图 ③ 状态迁移图 ④ 数据流程图

⑤ 时序图 ⑥ 事件追踪图 ⑦ 控制流程图 ⑧ 逻辑模拟图 ⑨ 仿真图 ⑩ 行为图

【6-12】在类的设计中需要遵循的方针是什么？三个主要的设计准则：抽象、信息隐蔽和模块化如何才能作到？

【6-13】在面向对象软件设计过程中，应按如下要求进行类的设计：只有类的共有界面的成员才能成为使用类的操作，这就是软件设计的(A)原则。当且仅当一个操作对类的实例的用户有用时，它才是类公共界面的一个成员，这是软件设计的(B)原则。由同属一个类的操作负担存取或加工类的数据，这是软件设计的(C)原则。两个类之间的交互应当仅涉及参数表，这是软件设计的(D)原则。每个派生类应该当做基类的特殊化来开发，而基类所具有的公共界面成为派生类的共有界面的一个子集，这是软件设计的(E)原则。

供选择的答案：

A：① 过程抽象 ② 功能抽象 ③ 信息隐蔽 ④ 共享性 ⑤ 连通性

B：① 标准调用 ② 最小界面 ③ 高耦合 ④ 高效率 ⑤ 可读性

C：① 数据抽象 ② 低内聚 ③ 高内聚 ④ 低复杂度 ⑤ 低强度

D：① 显式信息传递 ② 高内聚 ③ 低内聚 ④ 相互操作性 ⑤ 连接性

E： ① 动态联编 ② 异质表 ③ 信息隐蔽 ④ 多态性 ⑤ 继承性

【6-14】在类的通过复用的设计中，主要的继承关系有哪几种？试举例说明。

五、习题解答

【6-1】关于“面向对象”，有许多不同的看法。Coad 和 Yourdon 给出了一个定义：“面向对象 = 对象 + 类 + 继承 + 消息通信”。如果一个软件系统是使用这样 4 个概念设计和实现的，则认为这个软件系统是面向对象的。面向对象方法的特点是：

§ 方法的唯一性，即方法是对软件开发过程所有阶段进行综合考虑而得到的。

§ 从生存期的一个阶段到下一个阶段的高度连续性，即生存期后一阶段的成果只是在前一阶段成果的补充和修改。

§ 把面向对象分析(OOA)、面向对象设计(OOD)和面向对象程序设计(OOP)集成到生存期的相应阶段。

使用面向对象方法开发软件的好处是：

§ 开发方法的唯一性，开发阶段的高度连续性，表示方式的一致性；

§ 问题空间实体的自然表示，减轻了设计者的负担，在设计系统之初不必考虑一个很完整的解决方案。
§ 建立稳定的系统结构，可促进复用性，易于维护，易于修改，可合理利用共同性，减少复杂性。

【6-2】对象的定义：

- 对象是面向对象开发模式的基本成分，是现实世界中个体或事物的抽象表示。

- 每个对象可由一组属性和它可以执行的一组操作来定义。

可能的潜在对象有 7 类：

❶ 外部实体：它们产生或接受为目标系统所使用的信息。如各种物理设备、使用人员、其它相关的子系统。

❷ 事物：问题的信息域所涉及的概念实体。如各种报告、显示、文字、信号、规格说明等。

❸ 事件：系统运行时发生的并需要系统记忆的事件。如状态转换、物理运动等。

❹ 角色：与系统有交互的各种人员所扮演的角色。如经理、工程师、销售人员等。

❺ 场所或位置：建立系统整体环境或问题上下文的场所、位置。如基于计算机的系统的安装场所等。

❻ 组织机构：与应用有关的组织机构。如××组织，××部门等。

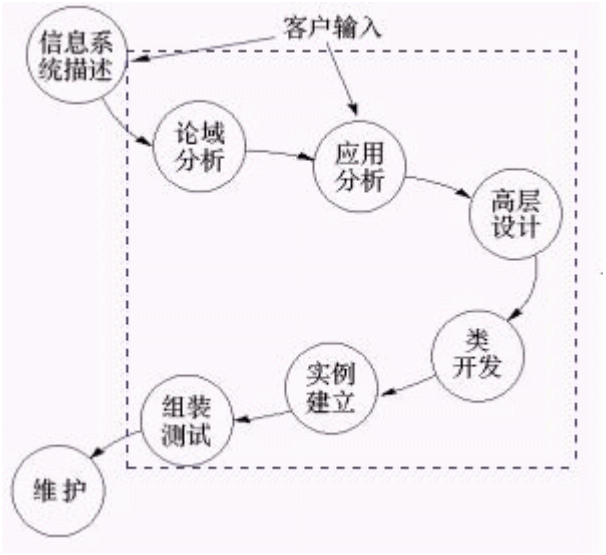
❼ 结构：定义由一组成分对象组成的聚合对象，或在极端情况下，定义对象的相关类。如传感器、四轮驱动车、计算机等。

6-3】把具有相同特征和行为的对象归在一起就形成了类。类成为某些对象的模板，抽象地描述了属于该类的全部对象的属性和操作。属于某个类的对象叫做该类的实例。对象的状态则包含在它的实例变量，即实例的属性中。类定义了各个实例所共有的结构，类的每一个实例都可以使用类中定义的操作。实例的当前状态是由实例所执行的操作定义的。

类，就它是一个数据值的聚合的意义上来看，与 Pascal 中的记录或 C 中的结构类似，但又有差别。类扩展了通常的记录语义，可提供各种级别的可访问性。也就是说，记录的某些成份可能是不可访问的，而这些成份对于本记录型来说具有可访问性。类不同于记录，因为它们包括了操作的定义，这些操作与类中声明的

数据值有相同的地位。

【6-4】基于复用的面向对象开发过程分为 6 个阶段，如右图中虚线框所围。



① 论域分析 ：论域分析开发问题论域的模型。论域分析应当在应用分析之前进行，我们在了解问题之前应当对问题敞开思想考虑，考察问题论域内的一个较宽的范围，分析覆盖的范围应比直接要解决的问题更多。

② 应用分析 ：应用(或系统)分析细化在论域分析阶段所开发出来的信息，并且把注意力集中于当前要解决的问题。因为通过论域分析，分析人员具有了较宽的论域知识，因而能开发出更好的抽象。

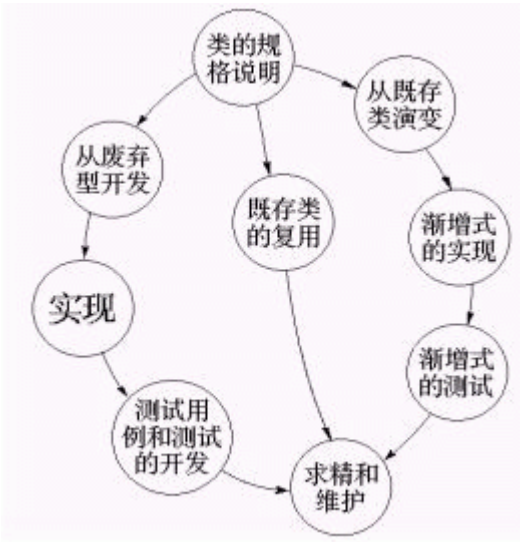
③ 高层设计 ：在一个纯面向对象环境中，软件体系结构设计与类设计常常是同样的过程，但还是应当把体系结构设计与类的设计分开。在高层设计阶段，设计应用系统的顶层视图。这相当于开发一个代表系统的类，通过建立该类的一个实例并发送一个消息给它来完成系统的“执行”。

④ 类的开发 ：根据高层设计所标识的对各个类的要求和类的规格说明，进行类的开发。因为一个应用系统往往是一个类的继承层次。对这些类的开发是最基本的设计活动。

⑤ 实例的建立：建立各个对象的实例，实现问题的解决方案。

⑥ 组装测试：按照类与类之间的关系组装一个完整的应用系统的过程中进行的测试。各个类的封装和类测试的完备性可减少组装测试所需要的时间。

【6-5】按照右图所示的类生存期，类的开发有三种方式。



(1) 既存类的复用

只要有可能就应复用既存类。为了达到此目的，开发人员必须能够找到这样一些类，它们都能选用来提供所需要的行为。有时应用要解决的问题与以前遇到的一些问题密切相关，因此那些问题中定义和实现的类可以复用。然而，多数照原样复用被限制在低层上最基本的类，像基本数据结构。对于较一般的结构，可以在实例化时，使用参数来规定它们的行为。

(2) 从既存类进行演化

多数复用情况是一个类已经存在，它提供的行为类似于要为新类定义的行为。开发人员可以使用既存类做为定义新类的起点。新类将根据既存类渐进式地演变而成。这样，在开发一个新类时，只需要花费较少的工作量就能复用许多既存类，得到所需要的新类。演化可以是横向的，也可以是纵向的。横向的演化导致既存类的一个新的版本，而纵向的演化将从既存类导出新类。我们在这里将主要讨论类的纵向渐进式开发。 ① 渐进式设计。设计既存类的一个特殊化类。通过确定新类中打算要的所有成员，设计者可以确定哪些追加的行为可以加到类中去，哪些既存的行为应当重新实现。

② 渐进式实现。许多实现可以从既存类直接继承；有时可以仅使用很少的新代码就能利用既存类的实现，而这些新代码必须当做老方法的上文或下文进行编写。此外，在渐进式设计阶段增加的那些行为也必须实现。

③ 渐进式测试。在测试中最花费时间的就是测试用例的生成。许多新类的测试用例可以从既存类的测试用例组中得到。新类的某些部分因为在测试既存类时已经测试过，因此可以不再需要测试。

(3) 从废弃型进行开发

这个分支仅在不得已的情况下使用。任何一个类，只要它的开发不涉及既存类，就可看做是一个新的继承结构的开始。因此，将建立两种类：一种是抽象类，它概括了将要表达的概念；另一种是具体类，它要实现这个概念。

① 设计。设计阶段需把分析阶段所产生的界面当做输入，并确定类的其它属性。设计给出类的所有细节。这个阶段的输出是有关类的属性的足够的细节，可支持它们的实现。单个类的设计包括构造数据存储，它是类定义的核心。其内部表示还包括一些私有函数，它们实现了共有操作的某些部分。单个类的低层设计还涉及一些重要联系，如继承和组装关系。

② 实现。通过变量的声明、操作界面的实现及支持界面操作的函数的实现，可实现一个类的预期行为和状态。在变量中存储的数据通常是其它类的实例，它们提供了为该类的开发所需的服务。

③ 测试。单个的类为测试提供了自然的单元。如果类的定义提供的界面比较狭窄，那么穷举测试就有可能实现。类的测试在最抽象的层次开始，沿继承联系继续向下进行，新的类可以很容易地完全地被测试，而已经测试过的部分就不需要从新测试了。

(4) 求精和维护

传统的维护活动是针对应用的，而求精过程则是针对类，并把类链接在一起的结构的。因为我们利用抽象进行开发，因此，维护部分在任一时间都能修改这些抽象。随着经验的增长，我们可以够标识抽象的抽象，使得继承结构通过泛化增加新的层次，即在既存的根类之上增加新的层次。

【6-6】结构化开发方法是使用最广泛、历史最长的过程化开发方法。结构化开发方法产生过程的抽象，这些抽象把软件视为处理流，定义构成一系列步骤的算法，每一步骤都是带有预定义输入和特定输出的一个过程，把这些步骤串联在一起可产生合理的稳定的贯通于整个程序的控制流。这将最终导致一个很简单的具有静态结构的体系结构。

在结构化开发方法中，数据结构是应算法步骤的要求而开发的。数据结构贯穿于过程，提供过程需要传送给它的操作的信息。系统的状态是一组全局变量，这组全局变量保持了状态的值，把它们从一个过程传送到另一个过程。

结构化开发方法是一种成熟的应用开发过程。对这种方法已经存在许多支持。然而，在大型系统的开发上和在面向用户系统的构造上存在一些问题。改进大型系统开发的技术主要集中在开发数据抽象。日益增多的考虑是使用抽象数据类型，把过程化系统开发过程包括到数据驱动的方法中。随着大型系统的开发，接踵而来的问题就是要把过程抽象与数据抽象方法组合起来，这种需要导致了面向对象开发方法的诞生。

面向对象开发方法是我们分解问题所使用方法演化的结果。在结构化开发方法中过程抽象是优先的，而面向对象开发方法中优先的是实体，即问题论域的对象。

在面向对象开发方法中，把标识和模型化问题论域中的主要实体做为系统开发的起点，主要考虑对象的行为而不是必须执行的一系列动作。

面向对象系统中的对象是数据抽象与过程抽象的一个混合体。表示这些实体的数据抽象是面向对象设计过程的主要产品，系统的状态保存在各个数据抽象的核心所定义的数据存储中。控制流被分成块，并被包括在各个在数据抽象上的各个操作里面。不像在结构化开发方法里那样，把数据从一个过程传送到另一个过程，而是控制流从一个数据抽象被传送到另一个数据抽象。完成的系统体系结构更复杂但也更灵活。在块中分离的控制流允许把复杂的动作视为局部的相互影响。

【6-7】A. ④ B. ① C. ③ D. ④ E. ① 其中，B、C、D 的答案可互换

面向对象的程序设计语言应具备面向对象方法所要求的 4 个成分：类、对象、继承和消息通信。类与对象由数据抽象和信息隐蔽得到，此外语言应具有信息继承的机制。 对象由一组属性和它可以执行的一组操作来定义。面向对象的软件系统通过对象间的消息通信和对象执行消息所要求的服务完成系统预定的功能。所以，对象在计算中可向其他对象发送消息，接受消息的对象必须通过响应消息／执行服务，给消息发送者以回答。

C++，Smalltalk，object C 是面向对象的程序设计语言，Ada、Modula2 是基于对象的程序设计语言，因为它缺少继承的机制，而 Pascal，C，APL 等都不是面向对象或基于对象的程序设计语言。

【6-8】论域分析是软件开发方法中一个基本组成部分，它给出的一组抽象是论域的知识的高层表示，用来做为特定系统需求开发的参考。由于分析常常是在超出当前应用的范围进行的，所以这个技术仅在扩充当前系统或将来建立其它系统时找寻复用信息时才是实用的。

论域分析是对与应用开发问题有关的范围十分广泛的知识的学习。论域的边界是模糊的，它们很多是凭借经验和实际考虑（如可用资源）定义的。主要思想是想把考虑的领域放宽一些，把相关的概念都标识到，以帮助更好地掌握应用的核心知识。当用户改变他们对系统需求的想法时，范围广泛的分析可以帮助预测这些变化，并不再需要进一步的分析。

论域分析的目的是标识基本概念，识别论域的特征，把这些概念集成到论域的模型中。这个模型必须包含概念之间的关系和关于每个单独概念的完全信息。这个信息起着一种胶合作用，把所有相关概念并入论域综合视图中去。

论域分析是一个持续的活动，它可以在软件开发的任一时刻进行标识，它还是一个可在超出特定应用的开发范围时仍然能够正常继续的处理。当论域变化时，必须更新那些抽象和关系。论域分析的结果使得我们可积累更多的有关论域的经验和细化我们的抽象。

使用论域抽象做为系统开发基础的好处是可适应性，客户可能改变需求，而且问题环境也可能改变。基于论域信息的应用更容易适应论域内知识的改变和用户需求的变化。

论域分析是一个长期的投资。如果在初始开发时需求发生变化，它可能有更直接的效益；在很大的项目上可能更快地得到回报。最大的价值是抽象的开发，这些抽象表示了一个问题论域中的基本概念，它们形成的软件库还可支持许多应用的开发。

应用(或系统)分析细化在论域分析阶段所开发出来的信息，并且把注意力集中于要解决的精确的问题。应当把客户对系统的需求当做限制来使用，它们缩减了论域的信息量，而这种信息将是主动设计过程的一部分。就这一点来说，保留的信息受到论域分析视野的影响。论域分析产生的模型并不需要用任何基于计算机系统的程序设计语言来表示，而应用分析阶段产生影响的条件则伴随着某种基于计算机系统的程序设计语言的表示。响应时间需求、用户界面需求和某些特殊的需求，如数据安全等，在这一层都被分解提取。

许多模型识别的要求是针对不止一个应用的。通常我们着重考虑两个方面：应用视图和类视图。必须对每个类的规格说明和操作详细化，还必须对形成应用结构的类之间的相互作用加以表示。

【6-9】 UML 叫做统一的建模语言，它把 Booch、Rumbaugh 和 Jacobson 等各自独立的 OOA 和 OOD 方法中最优秀的特色组合成一个统一的方法。UML 允许软件工程师使用由一组语法的语义的实用的规则支配的符号来表示分析模型。

在 UML 中用 5 种不同的视图来表示一个系统，这些视图从不同的侧面描述系统。每一个视图由一组图形来定义。这些视图概述如下：

§ 用户模型视图 ：这个视图从用户(在 UML 中叫做参与者)角度来表示系统。它用使用实例(use case)来建立模型，并用它来描述来自终端用户方面的可用的场景。

§ 结构模型视图 ：从系统内部来看数据和功能性。即对静态结构(类、对象和关系)模型化。

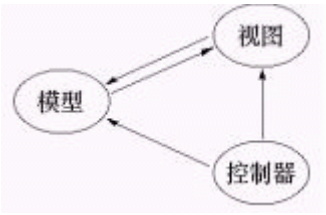
§ 行为模型视图 ：这种视图表示了系统动态和行为。它还描述了在用户模型视图和结构模型视图中所描述的各种结构元素之间的交互和协作。

§ 实现模型视图 ：将系统的结构和行为表达成为易于转换为实现的方式。

§ 环境模型视图 ：表示系统实现环境的结构和行为。

通常，UML 分析建模的注意力放在系统的用户模型和结构模型视图，而 UML 设计建模则定位在行为模型、实现模型和环境模型。

【6-10】一个典型的高层设计是在 Smalltalk 中使用的 MVC 软件体系结构，即模型 / 视图 / 控制器(Model / View / Controller)。



在这个结构中，模型是软件中的应用论域的各种对象，它们的操作独立于用户界面；视图则管理用户界面的输出；而控制器处理软件的输入。输入事件给出要发送给模型的消息。一旦模型改变了它的状态，就立即通过关联机制通知视图，让视图刷新显示。这个关联机制定义了在模型与各个视图之间的关系，它允许模型的运行独立于与它相关联的视图。类似地，控制器在输入事件发生时将对视图及模型进行控制与调度。 另一个典型的高层设计是 Coad 与 Yourdon 提出的 OOD 模型。这个模型也有 5 层结构，又被划分成了 4 个组成部分：问题论域、用户界面、任务管理和数据管理。



问题论域部分的职责是:组合所有的论域中定义的类,为应用设计适当的类层次,为适应语言要求简化继承层次，细化设计以改善性能，开发与数据管理部分的接口，做细化时加入低层对象，评审设计并审查对分析模型所做的增补。 人机交互部分的职责是：定义用户类型，开发任务场景，设计用户命令层次，建立原型以细化与用户交互，设计相关的类和类层次，适当时集成 GUI 类。

任务管理部分的职责是：标识任务的类型(如事件驱动，时钟驱动)，建立优先级，标识协调者，为每个任务设计适当的类。

数据管理部分的职责是：设计数据结构和布局，设计管理数据结构所需的服务，标识可以协助实现数据管理的工具，设计适当的类和类层次。

【6-11】A. ⑦ B. ⑥ C. ③ D. ⑥ E. ②

在 OMT 中，把分析时收集的信息建立在三个模型中。第一个模型是对象模型，它的作用是描述系统的静态结构，包括构成系统的对象和类，它们的属性和操作，以及它们之间的联系。第二个模型是动态模型，它描述系统的控制逻辑，主要涉及系统中各个对象和类的时序及变化状况。动态模型包括两种图， 即状态迁移图和事件追踪图。状态迁移图描述每一类对象的行为，事件追踪图描述发生于系

统执行过程中的某一特定场景。第三个模型是功能模型，它着重于描述系统内部数据的传送与处理，它由多个数据流图组成。

【6-12】在设计类时需要遵循的方针是：

§ 信息隐蔽：通过信息隐蔽可保护类的存储表示不被其它类的实例直接存取。

§ 消息限制：该类实例的用户应当只能使用界面提供的操作。

§ 狭窄界面：只有对其它类的实例是必要的操作才放到界面上。

§ 强内聚：模块内部各个部分之间应有较强的关系，它们不能分别标识。

§ 弱耦合：一个单独模块应尽量不依赖于其它模块。

§ 显式信息传递：两个类之间的交互应当仅涉及显式信息传递。

§ 派生类当做派生类型：每个派生类应该当做基类的特殊化来开发，而基类所具有的公共界面成为派生类的共有界面的一个子集。 § 抽象类：某些语言提供了一个类，用它做为继承结构的开始点，所有用户定义的类都直接或间接以这个类为基类。

为了在类的设计中做到抽象、信息隐蔽和模块化：

§ 以类作为系统的基本模块单元，通过一般化－特殊化关系和整体－部分关系，搭建整个系统的类层次结构，实现数据抽象和过程抽象；

§ 将数据和相关的操作封装在类内部，建立共有、私有和子类型等存取级别，将数据表示定义成为类的私有成员，实现信息隐蔽。

§ 通过建立类属性(类模板)，将某些有可复用要求的类设计成在数据类型上通用的可复用的软件构件，这样有助于实现模块化。

【6-13】A. ③ B. ② C. ③ D. ① E. ⑤

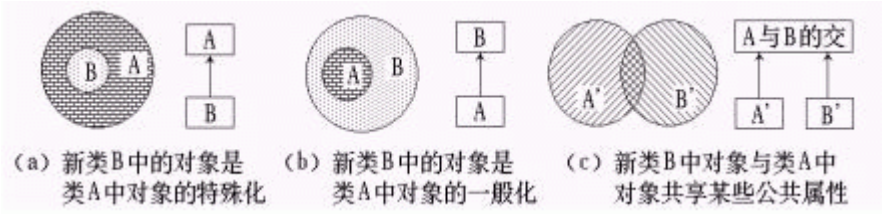
在面向对象软件设计过程中，应按如下要求进行类的设计：只有类的共有界面的成员才能成为使用类的操作，这就是软件设计的信息隐蔽原则。当且仅当一个操作对类的实例的用户有用时，它才是类公共界面的一个成员，这是软件设计的最小界面原则。由同属一个类的操作负担存取或加工类的数据，这是软件设计的高内聚原则。两个类之间的交互应当仅涉及参数表，这是软件设计的显式信息传递原则。每个派生类应该当做基类的特殊化来开发，而基类所具有的公共界面成为派生类的共有界面的一个子集，这是软件设计的继承性原则。

【6-14】在类的通过复用的设计中，主要的继承关系有两大类：

① 配置：利用既存类来设计类，可能会要求由既存类的实例提供类的某些特性。通过把相应类的实例声明为新类的属性来配置新类。例如，一种仿真服务器可能要求使用一个计时器来跟踪服务时间。设计者不必开发在这个行为中所需的数据和操作，而是应当找到计时器类，并在服务器类的定义中声明它。

但如果使用既存类的内部表示来做为新类的内部表示的一部分，这是一种“针对实现”的继承方式，这种继承方式不好。例如，考虑使用继承来实现一个 Circle 类。Point 类可支持 Circle 类的一部分实现。为了定义一个圆，我们只需要定义一个点和一个值，做为圆的圆心和半径。把 Point 当做子类，Circle 类不但能得到由 x 和 y 提供的圆心，而且还能得到一个操作，让圆能够自由移动。但这样做，我们失去了抽象。

② 演变 ：要开发的新类可能与一个既存类非常类似，但不完全相同。此时可以从一个既存类演变成一个新类，可以利用继承机制来表示一般化－特殊化的关系。特殊化处理有三种可能的方式。



§ 如果新的概念是一个既存类所表示概念的一个特殊情况，特殊化运算可以从该既存类的定义产生新类的初始构造，这是典型的类继承的使用。既存类 A 的数据结构和操作可以成为新类 B 的一部分，如图(a)所示。既存类 A 的公共操作成为新类 B 的共有界面部分。
§ 如果新类比软件库中那些既存类更一般，则新类 B 不具有既存类 A 的全部特性，一般化运算把两个类中共同的特性移到新的更高层的类中，高一层的类是 B，我们将要设计它。原来的类 A 成为新类 B 的子类。如图(b)所示。

§ 一个既存类 A 与我们设计的新类 B 共享概念的某一个部分，则两个概念的共同部分形成新类的基础，且既存类与新类两者成为子类，如图(c)所示。

第七章 软件维护

一、复习要求

1. 了解软件质量定义和软件质量度量。

2. 了解软件维护的类型与策略。

3. 了解软件维护的过程与管理方法。

4. 了解可维护性的概念。

5. 了解提高可维护性的方法。

6. 了解软件逆向工程与再工程的概念

二、内容提要

1. 软件质量的概念

(1) 软件质量的定义

关于软件质量的定义，曾给出过多种定义。

§ ANSI/IEEE Std 729-1983 定义软件质量为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。

§ M.J. Fisher 定义软件质量为“所有描述计算机软件优秀程度的特性的组合”。

也就是说，为满足软件的各项精确定义的功能、性能需求，符合文档化的开发标准，需要相应地给出或设计一些质量特性及其组合，作为在软件开发与维护中的重要考虑因素。如果这些质量特性及其组合都能在产品中得到满足，则这个软件产品质量就是高的。

软件质量反映了以下三方面的问题：

§ 软件需求是度量软件质量的基础。不符合需求的软件就不具备质量。

§ 规范化的标准定义了一组开发准则，用来指导软件人员用工程化的方法来开发软件。如果不遵守这些开发准则，软件质量就得不到保证。

§ 往往会有一些隐含的需求没有显式地提出来。如软件应具备良好的可维护性。如果软件只满足那些精确定义了的需求而没有满足这些隐含的需求，软件质量也不能保证。 软件质量是各种特性的复杂组合。它随着应用的不同而不同，随着用户提出的质量要求不同而不同。因此，有必要讨论各种质量特性，以及评价质量的准则，还要介绍为保证质量所进行的各种活动。

(2) 软件质量特性与质量模型

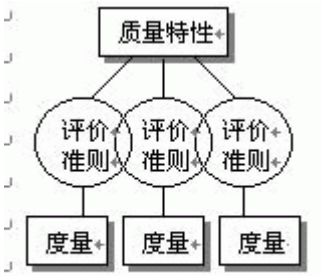


图 7.1 McCall 质量模型框架

软件质量特性，反映了软件的本质。讨论一个软件的质量，问题最终要归结到定义软件的质量特性。而定义一个软件的质量，就等价于为该软件定义一系列质量特性。

人们通常把影响软件质量的特性用软件质量模型来描述。已有多种有关软件质量模型的方案。它们共同的特点是：把软件质量特性定义成分层模型。最基本的叫做基本质量特性，它可以由一些子质量特性定义和度量。子质量特性在必要时又可由它的一些子质量特性定义和度量。

早在 1976 年，由 Boehm 等提出软件质量模型的分层方案。1979 年 McCall 等人改进 Boehm 质量模型又提出了一种软件质量模型。模型的三层次式框架如图 7.1 所示。质量模型中的质量概念基于 11 个特性之上。而这 11 个特性分别面向软件产品的运行、修正、转移。它们与特性的关系如图 7.2 所示。McCall 等认为，特性是软件质量的反映，软件属性可用做评价准则，定量化地度量软件属性可知软件质量的优劣。



图 7.2 McCall 软件质量模型

McCall 等人的质量特性定义如下：

正确性	在预定环境下，软件满足设计规格说明及用户预期目标的程度。它要求软件本身没有错误。
可靠性	软件按照设计要求，在规定时间和条件下不出故障，持续运行的程度。
效率	为了完成预定功能，软件系统所需的计算机资源的多少。
完整性	为某一目的而保护数据，避免它受到偶然的或有意的破坏、改动或遗失的能力。
可使用性	对于一个软件系统，用户学习、使用软件及为程序准备输入和解释输出所需工作量的大小。
可维护性	为满足用户新的要求，或当环境发生了变化，或运行中发现了新的错误时，对一个已投入运行的软件进行相应诊断和修改所需工作量的大小。
可测试性	测试软件以确保其能够执行预定功能所需工作量的大小。
灵活性	修改或改进一个已投入运行的软件所需工作量的大小。
可移植性	将一个软件系统从一个计算机系统或环境移植到另一个计算机系统或环境中运行时所需工作量的大小。
可复用性	一个软件(或软件的部件)能再次用于其它应用(该应用的功能与此软件或软件部件的所完成的功能有关)的程度。
互连性	又称相互操作性。连接一个软件和其它系统所需工作量的大小。如果这个软件要联网或与其它系统通信或要把其它系统纳入到自己的控制之下，必须有系统间的接口，使之可以联结。

对以上各个质量特性直接进行度量是很困难的，有些情况下甚至是不可能的。因此，McCall 定义了一些评价准则，使用它们对反映质量特性的软件属性分级，以此来估计软件质量特性的值。软件属性一般分级范围从 0 (最低)到 10 (最高)。

各评价准则定义如下。

可跟踪性	在特定的开发和运行环境下跟踪设计表示或实际程序部件到原始需求的(可追溯)能力。
完备性	软件需求充分实现的程度。
一致性	在整个软件设计与实现的过程中技术与记号的统一程度。
安全性	防止软件受到意外的或蓄意的存取、使用、修改、毁坏，或防止泄密的程度。
容错性	系统出错(机器临时发生故障或数据输入不合理)时，能以某种预定方式，做出适当处理，得以继续执行和恢复系统的能力。它又称健壮性。
准确性	能达到的计算或控制精度。它又称精确性。
简单性	在不复杂、可理解的方式下，定义和实现软件功能的程度。
执行效率	为了实现某个功能，提供使用最少处理时间的程度。
存储效率	为了实现某个功能，提供使用最少存储空间的程度。
存取控制	软件对用户存取权限的控制方式达到的程度。
存取审查	软件对用户存取权限的检查程度。
操作性	操作软件的难易程度。它通常取决于与软件操作有关的操作规程，以及是否提供有用的输入／输出方法。
易训练性	软件辅助新的用户使用系统的能力。这取决于是否提供帮助用户熟练掌握软件系统的方法。它又称可培训性或培训性。

简明性	软件易读的程度。这个特性可以帮助人们方便地阅读本人或他人编制的程序和文档。它又称可理解性。
模块独立性	软件系统内部接口达到的高内聚、低耦合的程度。
自描述性	对软件功能进行自我说明的程度。亦称自含文档性。
结构性	软件能达到的结构良好的程度。
文档完备性	软件文档齐全、描述清楚、满足规范或标准的程度。
通用性	软件功能覆盖面宽广的程度。
可扩充性	软件的体系结构、数据设计和过程设计的可扩充的程度。
可修改性	软件容易修改，而不致于产生副作用的程度。
自检性	软件监测自身操作效果和发现自身错误的能力，又称工具性。
机器独立性	不依赖于某个特定设备及计算机而能工作的程度，又称硬件独立性。
软件独立性	软件不依赖于非标准程序设计语言特征、操作系统特征，或其它环境约束，仅靠自身能实现其功能的程度，又称自包含性。
通信共享性	使用标准的通信协议、接口和带宽的标准化的程度。
数据共享性	使用标准数据结构和数据类型的程度。
通信性	提供有效的I／O方式的程度。

正确性和容错性是相互补充的。正确的程序不一定是可容错的程序。反过来，可容错的程序不一定是完全正确的程序。我们要求一个可靠的软件应当在正常的情况下能够正确地工作；而在意外的情况下，也能做出适当的处理，隔离故障，尽快地恢复。这才是一个好的程序。此外，有人在灵活性中加了一个评价准则，叫做“可重配置特性”，它是指软件系统本身各部分的配置能按用户要求实现的容易程度。在简明性中也加了一个评价准则，即“清晰性”，它是指软件的内部结构、内部接口要清晰，人－机界面要清晰。

(3) ISO 的软件质量评价模型

ISO/IEC 9126-1991 标准规定的软件质量模型由三层组成。在这个标准中，三层次中的第一层为称为质量特性，第二层称为质量子特性，第三层称为度量。如图 7.3 所示。该标准定义了 6 个质量特性，即功能性、可靠性、可维护性、效率、可使用性、可移植性；并推荐了 21 个子特性，如适合性、准确性、互操作性、依从性、安全性、成熟性、容错性、易恢复性、易理解性、易学习性、易操作性、时间特性、资源特性、易分析性、易变更性、稳定性、易测试性、适应性、易安装性、遵循性、易替换性，但不做为标准。用于评价质量子特性的度量没有统一的标准，由各使用单位视实际情况制定。

(4) 软件质量特性之间的竞争

在软件的质量特性与质量特性之间、质量特性与子特性之间存在着有利影响和不利影响，若用“△”表示该质量特性对质量特性有有利影响；用“▽”表示该质量特性对质量特性有不利影响。 则有下面表 7.1 所示的关系。例如，由于效率的要求，应尽可能采用汇编语言。但是用汇编语言编制出的程序，可靠性、可移植性以及可维护性都很差。 在进行软件质量设计时，必须考虑利弊，全面权衡，根据质量需求，适当合理地选择 / 设计质量特性，并进行评价。

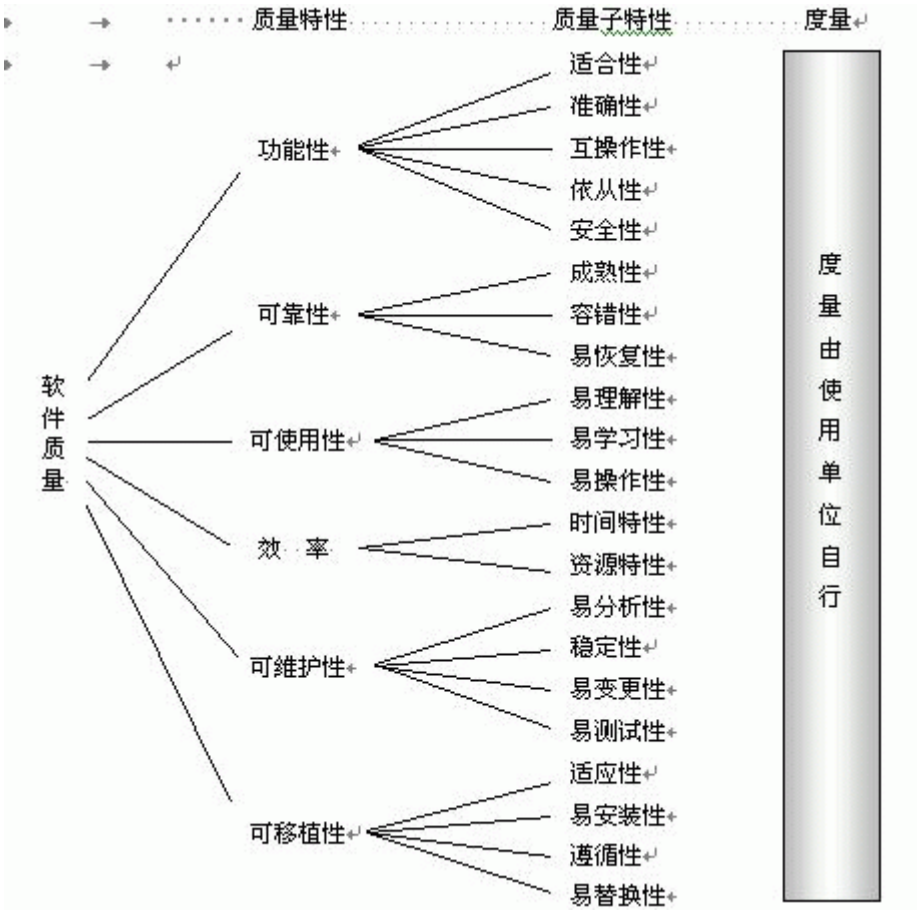


图 7.3 ISO 软件质量度量模型

表 7.1 各质量特性与质量特性之间的关系

	功能性	可靠性	可使用性	效率	可维护性	可移植性
功能性		△			△	
可靠性				▽		△
可使用性				▽	△	△
效率		▽			▽	▽
可维护性		△		▽		△
可移植性		▽		▽		

(5) 质量活动

产品的高质量将导致成本的下降。质量活动开始于 20 世纪 40 年代 E. Edwards Deming 的开创性工作，由日本人发扬广大，他们开发了一种系统化的方法以从根本上消除造成产品缺陷的原因。1970 年代到 1980 年代，他们的工作移植到西方，称为“全面质量管理(TQC)”。该活动采用的是 4 个步骤的过程：

① 开展：是一个连续的过程改进体系，其目的是开发一个看得见的可重复的和可度量的过程；

② 当然的质量：只有在前一步完成之后才可启动。这一步将检查影响过程的无形因素，并优化这些因素对过程的影响。例如，软件过程可能受到高层职员流动的影响，而这又是由于公司内部不断重组引起的。公司组织的稳定对于软件质量的提高有很大帮助，因此在这一步可以对公司的重组提出建议。

③ 感性：意指“第五感觉”。这一步从过程转到用户身上。通过检查用户使用软件产品的情况，改进产品本身，并(潜在地)改进软件产品的生产过程。

④ 有魅力的质量：通过观察产品在市场上的使用，寻找产品在相关领域发展的方向，从而开发出有预见性的新产品。

2. 软件维护的概念

(1) 软件维护的定义

我们称在软件运行／维护阶段对软件产品所进行的修改就是所谓的维护。要求进行维护的原因多种多样，归结起来有三种类型：

§ 改正在特定的使用条件下暴露出来的一些潜在程序错误或设计缺陷；

§ 因在软件使用过程中数据环境发生变化或处理环境发生变化，需要修改软件以适应这种变化。

§ 用户和数据处理人员在使用时常提出改进现有功能，增加新的功能，以及改善总体性能的要求，为满足这些要求，就需要修改软件把这些要求纳入到软件之中。

由这些原因引起的维护活动可以归为以下几类：

① 改正性维护

在软件交付使用后，必然会有一部分隐藏的错误被带到运行阶段来。这些隐藏下来的错误在某些特定的使用环境下就会暴露出来。为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，应当进行的诊断和改正错误的过程，就叫做改正性维护。

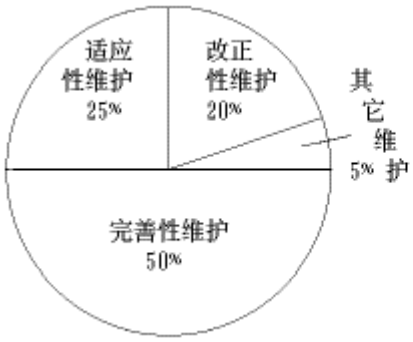
② 适应性维护

随着计算机的飞速发展，外部环境(新的硬、软件配置)或数据环境(数据库、数据格式、数据输入／输出方式、数据存储介质)可能发生变化，为了使软件适应这种变化，而去修改软件的过程就叫做适应性维护。

③ 完善性维护

在软件的使用过程中，用户往往会对软件提出新的功能与性能要求。为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效

率、提高软件的可维护性。这种情况下进行的维护活动叫做完善性维护。 在维护阶段的最初一、二年，改正性维护的工作量较大。随着错误发现率急剧降低，并趋于稳定，就进入了正常使用期。然而，由于改造的要求，适应性维护和完善性维护的工作量逐步增加。实践表明，在几种维护活动中，完善性维护所占的比重最大，来自用户要求扩充、加强软件功能、性能的维护活动约占整个维护工作的 50%。



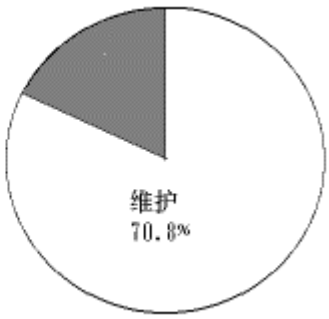


图 7.4 三类维护占总维护比例 图 7.5 维护在软件生存期所占比例

④ 预防性维护

除了以上三类维护之外，还有一类维护活动，叫做预防性维护。这是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。通常，预防性维护定义为：“把今天的方法学用于昨天的系统以满足明天的需要”。也就是说，采用先进的软件工程方法对需要维护的软件或软件中的某一部分(重新)进行设计、编制和测试。
在整个软件维护阶段所花费的全部工作量中，预防性维护只占很小的比例，而完善性维护占了几乎一半的工作量。参看图 7.4。从图 7.5 中可以看到，软件维护活动所花费的工作占整个生存期工作量的 70%以上。

(2) 影响维护工作量的因素

在软件维护中，影响维护工作量的程序特性有以下 6 种。

§ 系统大小：系统越大，理解掌握起来越困难。系统越大，所执行功能越复杂。因而需要更多的维护工作量。

§ 程序设计语言：语言的功能越强，生成程序所需的指令数就越少；语言的功能越弱，实现同样功能所需语句就越多，程序就越大。有许多软件是用较老的程序设计语言书写的，程序逻辑复杂而混乱，且没有做到模块化和结构化，直接影响到程序的可读性。

§ 系统年龄：老系统随着不断的修改，结构越来越乱；由于维护人员经常更换，程序又变得越来越难于理解。而且许多老系统在当初并未按照软件工程的要求进行开发，因而没有文档，或文档太少，或在长期的维护过程中文档在许多地方与程序实现变得不一致，这样在维护时就会遇到很大困难。

§ 数据库技术的应用：使用数据库，可以简单而有效地管理和存储用户程序中的数据，还可以减少生成用户报表应用软件的维护工作量。

§ 先进的软件开发技术：在软件开发时，若使用能使软件结构比较稳定的分析与设计技术，及程序设计技术，如面向对象技术、复用技术等，可减少大量的工作量。

§ 其它：例如，应用的类型、数学模型、任务的难度、开关与标记、IF 嵌套深度、索引或下标数等，对维护工作量都有影响。

此外，许多软件在开发时并未考虑将来的修改，这就为软件的维护带来许多问题。

(3) 软件维护的策略

根据影响软件维护工作量的各种因素，针对三种典型的维护，James Martin 等提出了一些策略，以控制维护成本。

① 改正性维护

要生成 100％可靠的软件成本太高，不一定合算。但通过使用新技术，可大大提高可靠性，减少进行改正性维护的需要。这些技术包括：数据库管理系统、软件开发环境、程序自动生成系统、较高级(第四代)的语言，应用以上 4 种方法可产生更加可靠的代码。此外，

§ 利用应用软件包，可开发出比由用户完全自己开发的系统可靠性更高的软件。

§ 结构化技术，用它开发的软件易于理解和测试。

§ 防错性程序设计。把自检能力引入程序，通过非正常状态的检查，提供审查跟踪。

§ 通过周期性维护审查，在形成维护问题之前就可确定质量缺陷。

② 适应性维护

这一类的维护不可避免，但可以控制。

§ 在配置管理时，把硬件、操作系统和其它相关环境因素的可能变化考虑在内，可以减少某些适应性维护的工作量。

§ 把与硬件、操作系统，以及其它外围设备有关的程序归到特定的程序模块中。可把因环境变化而必须修改的程序局部于某些程序模块之中。

§ 使用内部程序列表、外部文件，以及处理的例行程序包，可为维护时修改程序提供方便。

③ 完善性维护

利用前两类维护中列举的方法，也可以减少这一类维护。特别是数据库管理系统、程序生成器、应用软件包，可减少系统或程序员的维护工作量。

此外，建立软件系统的原型，把它在实际系统开发之前提供给用户。用户通过研究原型，进一步完善他们的功能要求，就可以减少以后完善性维护的需要。

(4) 维护成本

有形的软件维护成本是花费了多少钱，而其它非直接的的维护成本有更大的影响。例如，无形的成本可以是：

§ 一些看起来是合理的修复或修改请求不能及时安排，使得客户不满意；

§ 变更的结果把一些潜在的错误引入正在维护的软件，使得软件整体质量下降；

§ 当必须把软件人员抽调到维护工作中去时，就使得软件开发工作受到干扰。

软件维护的代价是在生产率方面的惊人下降。有报告说，生产率将降到原来的40分之一。维护工作量可以分成生产性活动(如分析和评价、设计修改和实现)和“轮转”活动(如力图理解代码在做什么、试图判明数据结构、接口特性、性能界限等)。下面的公式给出了一个维护工作量的模型：

$$M = p + Ke^{c-d}$$

其中，M是维护中消耗的总工作量，p是上面描述的生产性工作量，K是一个经验常数，c是因缺乏好的设计和文档而导致复杂性的度量，d是对软件熟悉程度的度量。

这个模型指明，如果使用了不好的软件开发方法(未按软件工程要求做)，原来参加开发的人员或小组不能参加维护，则工作量(及成本)将按指数级增加。

3. 软件维护活动

(1) 维护机构

除了较大的软件开发公司外，通常在软件维护工作方面，不保持正式的维护机构。维护往往是在没有计划的情况下进行的。虽然不要求建立一个正式的维护机构，但是在开发部门，确立一个非正式的维护机构则是非常必要的。例如图7.6就是一个维护机构的组织方案。

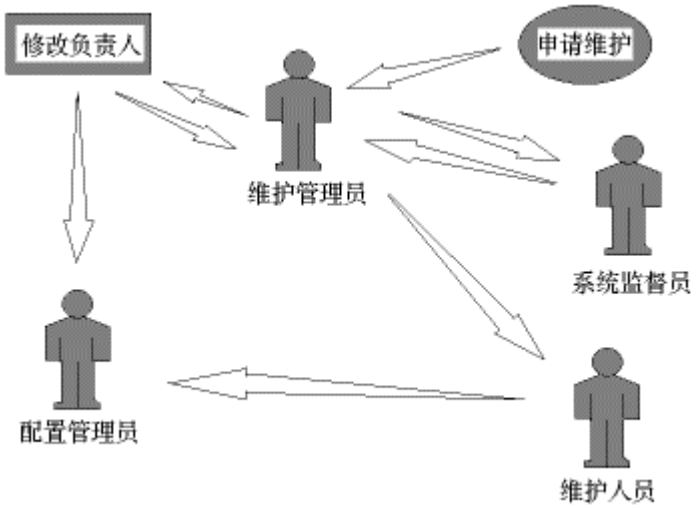


图 7.6 软件维护的机构

维护申请提交给一个维护管理员，他把申请交给某个系统监督员去评价。一旦做出评价，由修改负责人确定如何进行修改。在维护人员对程序进行修改的过程中，由配置管理员严格把关，控制修改的范围，对软件配置进行审计。

维护管理员、系统监督员、修改负责人等，均代表维护工作的某个职责范围。修改负责人、维护管理员可以是指定的某个人，也可以是一个包括管理人员、高级技术人员在内的小组。系统监督员可以有其他职责，但应具体分管某一个软件包。在开始维护之前，就把责任明确下来，可以大大减少维护过程中的混乱。

(2) 软件维护工作流程

§ 先确认维护要求。这需要维护人员与用户反复协商，弄清错误概况以及对业务的影响大小，以及用户希望做什么样的修改。然后由维护组织管理员确认维护类型。

§ 对于改正性维护申请，从评价错误的严重性开始。如果存在严重的错误，则必须安排人员，在系统监督员的指导下，进行问题分析，寻找错误发生的原因，进行“救火”性的紧急维护；对于不严重的错误，可根据任务、机时情况、视轻重缓急，进行排队，统一安排时间。

§ 对于适应性维护和完善性维护申请，需要先确定每项申请的优先次序。若某项申请的优先级非常高，就可立即开始维护工作，否则，维护申请和其它的开发工作一样，进行排队，统一安排时间。

§ 尽管维护申请的类型不同，但都要进行同样的技术工作。这些工作有：修改软件需求说明、修改软件设计、设计评审、对源程序做必要的修改、单元测试、集成测试(回归测试)、确认测试、软件配置评审等。

§ 在每次软件维护任务完成后，最好进行一次情况评审，确认：在目前情况下，设计、编码、测试中的哪一方面可以改进？哪些维护资源应该有但没有？工作中主要的或次要的障碍是什么？从维护申请的类型来看是否应当有预防性维护？

(3) 维护评价

评价维护活动比较困难，因为缺乏可靠的数据。但如果维护记录做得比较好，就可以得出一些维护“性能”方面的度量值。可参考的度量值如：

§ 每次程序运行时的平均出错次数；

§ 花费在每类维护上的总“人时”数；

§ 每个程序、每种语言、每种维护类型的程序平均修改次数；

§ 因为维护，增加或删除每个源程序语句所花费的平均“人时”数；

§ 用于每种语言的平均“人时”数；

§ 维护申请报告的平均处理时间；

§ 各类维护申请的百分比。

这七种度量值提供了定量的数据，据此可对开发技术、语言选择、维护工作计划、资源分配、以及其它许多方面做出判定。因此，这些数据可以用来评价维护工作。

4. 程序修改的步骤及修改的副作用

(1) 分析和理解程序

经过分析，全面、准确、迅速地理解程序是决定维护成败和质量好坏的关键。在这方面，软件的可理解性和文档的质量非常重要。必须：

§ 理解程序的功能和目标；

§ 掌握程序的结构信息，即从程序中细分出若干结构成分。如程序系统结构、控制结构、数据结构和输入／输出结构等；

§ 了解数据流信息，即所涉及到的数据来源何处，在哪里被使用；

§ 了解控制流信息，即执行每条路径的结果；

§ 理解程序的操作(使用)要求；

为了容易地理解程序，要求自顶向下地理解现有源程序的程序结构和数据结构，为此可采用如下方法：

① 分析程序结构图：

§ 分析各个过程的源代码，建立一个直接调用矩阵 D 或调用树。

§ 建立过程的间接调用矩阵。

§ 分析各个过程的接口，估计更改的复杂性。

② 数据跟踪

§ 建立各层次的程序级上的接口图，展示各模块或过程的调用方式和接口参数；

§ 利用数据流分析方法，对过程内部的一些变量进行跟踪；维护人员通过这种数据流跟踪，可获得有关数据在过程间如何传递，在过程内如何处理等信息。

③ 控制跟踪

控制流跟踪同样可在结构图基础上或源程序基础上进行。可采用符号执行或实际动态跟踪的方法，了解数据如何从一个输入源到达输出点的。

④ 在分析的过程中，充分阅读和使用源程序清单和文档，分析现有文档的合理性。

⑤ 充分使用由编译程序或汇编程序提供的交叉引用表、符号表、以及其它有用的信息。

⑥ 如有可能，积极参加开发工作。

(2) 修改程序

对程序的修改，必须事先做出计划，有预谋地、周密有效地实施修改。

① 设计程序的修改计划

程序的修改计划要考虑人员和资源的安排。修改计划的内容主要包括： § 规格说明信息：数据修改、处理修改、作业控制语言修改、系统之间接口的修改等；

§ 维护资源：新程序版本、测试数据、所需的软件系统、计算机时间等；

§ 人员：程序员、用户相关人员、技术支持人员、厂家联系人、数据录入员等；

§ 提供：纸面、计算机媒体等。

针对以上每一项，要说明必要性、从何处着手、是否接受、日期等。通常，可采用自顶向下的方法，在理解程序的基础上，

ⅰ）研究程序的各个模块、模块的接口、及数据库，从全局的观点，提出修改计划。

ii）依次地把要修改的、以及那些受修改影响的模块和数据结构分离出来。为此，要

§ 识别受修改影响的数据；

§ 识别使用这些数据的程序模块；

§ 对于上面程序模块，按是产生数据、修改数据、还是删除数据进行分类；

§ 识别对这些数据元素的外部控制信息；

§ 识别编辑和检查这些数据元素的地方；

§ 隔离要修改的部分；

iii）详细地分析要修改的、以及那些受变更影响的模块和数据结构的内部细节，设计修改计划，标明新逻辑及要改动的现有逻辑。

iv）向用户提供回避措施。用户的某些业务因软件中发生问题而中断，为不让系统长时间停止运行，需把问题局部化，在可能的范围内继续开展业务。可以采取的措施有：

§ 在问题的原因还未找到时，先就问题的现象，提供回避的操作方法。

§ 如果弄清了问题的原因，可通过临时修改或改变运行控制以回避在系统运行时产生的问题。

② 修改代码，以适应变化

§ 正确、有效地编写修改代码；

§ 要谨慎地修改程序，尽量保持程序的风格及格式，要在程序清单上注明改动的指令；

§ 不要删除程序语句，除非完全肯定它是无用的；

§ 不要试图共用程序中已有的临时变量或工作区，为了避免冲突或混淆用途，应自行设置自己的变量；

§ 插入错误检测语句；

§ 在修改过程中做好修改的详细记录，消除变更中任何有害的副作用(波动效应）；

③ 修改程序的副作用

所谓副作用是指因修改软件而造成的错误或其它不希望发生的情况，有三种副作用：

§ 修改代码的副作用。在使用程序设计语言修改源代码时，都可能引入错误。例如，删除或修改一个子程序、删除或修改一个标号、删除或修改一个标识符、改变程序代码的时序关系、改变占用存储的大小、改变逻辑运算符、修改文件的打开或关闭、改进程序的执行效率，以及把设计上的改变翻译成代码的改变、为边界条件的逻辑测试做出改变时，都容易引入错误。

§ 修改数据的副作用。在修改数据结构时，有可能造成软件设计与数据结构不匹配，因而导致软件出错。数据副作用就是修改软件信息结构导致的结果。例如，在重新定义局部的或全局的常量、重新定义记录或文件的格式、增大或减小一个数组或高层数据结构的大小、修改全局或公共数据、重新初始化控制标志或指针、重新排列输入／输出或子程序的参数时，容易导致设计与数据不相容的错误。数据副作用可以通过详细的设计文档加以控制。在此文档中描述了一种交叉引用，把数据元素、记录、文件和其它结构联系起来。

§ 文档的副作用。对数据流、软件结构、模块逻辑或任何其它有关特性进行修改时，必须对相关技术文档进行相应修改。否则会导致文档与程序功能不匹配，缺省条件改变，新错误信息不正确等错误。使得软件文档不能反映软件的当前状态。对于用户来说，软件事实上就是文档。如果对可执行软件的修改不反映在文档里，就会产生文档的副作用。例如，对交互输入的顺序或格式进行修改，如果没有正确地记入文档中，就可能引起重大的问题。过时的文档内容、索引和文本可能造成冲突，引起用户的失败和不满。因此，必须在软件交付之前对整个软件配置进行评审，以减少文档的副作用。 为了控制因修改而引起的副作用，要做到：

ⅰ）按模块把修改分组；

ⅱ）自顶向下地安排被修改模块的顺序；

ⅲ）每次修改一个模块；

ⅳ）对于每个修改了的模块，在安排修改下一个模块之前，要确定这个修改的副作用。可以使用交叉引用表、存储映象表、执行流程跟踪等。

(3) 重新验证程序

在将修改后的程序提交用户之前，需要用以下的方法进行充分的确认和测试，以保证整个修改后的程序的正确性。

① 静态确认

修改软件，伴随着引起新的错误的危险。为了能够做出正确的判断，验证修改后的程序至少需要两个人参加。要检查

§ 修改是否涉及到规格说明？修改结果是否符合规格说明？有没有歪曲规格说明？

§ 程序的修改是否足以修正软件中的问题？源程序代码有无逻辑错误？修改时有无修补失误？

§ 修改部分对其它部分有无不良影响(副作用)？

对软件进行修改，常常会引发别的问题，因此有必要检查修改的影响范围。

② 计算机确认

在充分进行了以上确认的基础上，要用计算机对修改程序进行确认测试：

§ 确认测试顺序：先对修改部分进行测试，然后隔离修改部分，测试程序的未修改部分，最后再把它们集成起来进行测试。这种测试称为回归测试。

§ 准备标准的测试用例。

§ 充分利用软件工具帮助重新验证过程。

§ 在重新确认过程中，需邀请用户参加。

③ 维护后的验收——在交付新软件之前，维护主管部门要检验：

§ 全部文档是否完备，并已更新；

§ 所有测试用例和测试结果已经正确记载；

§ 记录软件配置所有副本的工作已经完成；

§ 维护工序和责任已经确定。

从维护角度来看所需测试种类如：

- 对修改事务的测试；
- 对修改程序的测试；

- 操作过程的测试；
- 应用系统运用过程的测试；

- 使用过程的测试；
- 系统各部分之间接口的测试；

- 作业控制语言的测试；
- 与系统软件接口的测试；

- 软件系统之间接口的测试；
- 安全性测试；

- 后备 / 恢复过程的测试。

5. 软件可维护性

(1) 软件可维护性的定义

所谓软件可维护性，是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充或压缩的容易程度。可维护性、可使用性、可靠性是衡量软件质量的几个主要质量特性，也是用户十分关心的几个方面。可惜的是影响软件质量的这些重要因素，目前尚没有对它们定量度量的普遍适用的方法。但是就它们的概念和内涵来说则是很明确的。

软件的可维护性是软件开发阶段各个时期的关键目标。

目前广泛使用的是用如下的七个特性来衡量程序的可维护性。而且对于不同类型的维护，这七种特性的侧重点也不相同。表 7.2 显示了在各类维护中应侧重哪些特性。图中的“○”表示需要的特性。

表 7.2 在各类维护中的侧重点

	改正性维护	适应性维护	完善性维护
可理解性	√		
可测试性	√		
可修改性	√	√	
可靠性	√		
可移植性		√	
可使用性		√	√
效率			√

上面所列举的这些质量特性通常体现在软件产品的许多方面，为使每一个质量特性都达到预定的要求，需要在软件开发的各个阶段采取相应的措施加以保证。因此，软件的可维护性是产品投入运行以前各阶段面向上述各质量特性要求进行开发的最终结果。

(2) 可维护性的度量

人们一直期望对软件的可维护性做出定量度量，但要做到这一点并不容易。许多研究工作集中在这个方面，形成了一个引人注目的学科——软件度量学。下面我

们介绍度量一个可维护的程序的七种特性时常用的方法。这就是质量检查表、质量测试、质量标准。
质量检查表是用于测试程序中某些质量特性是否存在的一个问题清单。评价者针对检查表上的每一个问题，依据自己的定性判断，回答“Yes”或者“No”。质量测试与质量标准则用于定量分析和评价程序的质量。由于许多质量特性是相互抵触的，要考虑几种不同的度量标准，相应地去度量不同的质量特性。

① 可理解性

可理解性表明人们通过阅读源代码和相关文档，了解程序功能及其如何运行的容易程度。一个可理解的程序主要应具备以下一些特性：模块化(模块结构良好、功能完整、简明)，风格一致性(代码风格及设计风格的一致性)，不使用令人捉摸不定或含糊不清的代码，使用有意义的数据名和过程名，结构化，完整性(对输入数据进行完整性检查)等。

用于可理解性度量的检查表的内容有：程序是否模块化？结构是否良好？每个模块是否有注释块，说明程序的功能、主要变量的用途及取值、所有调用它的模块、以及它调用的所有模块？在模块中是否有其它有用的注释内容，包括输入输出、精确度检查、限制范围和约束条件、假设、错误信息、程序履历等？在整个程序中缩进和间隔的使用风格是否一致？在程序中每一个变量，过程是否具有单一的有意义的名字？程序是否体现了设计思想？程序是否限制使用一般系统中没有的内部函数过程与子程序？是否能通过建立公共模块或子程序来避免多余的代码？所有变量是否是必不可少的？是否避免了把程序分解成过多的模块、函数或子程序？程序是否避免了很难理解的、非标准的语言特性？

对于可理解性，可以使用一种叫做“90－10 测试”的方法来衡量。即把一份被测试的源程序清单拿给一位有经验的程序员阅读 10 分钟，然后把这个源程序清单拿开，让这位程序员凭自己的理解和记忆，写出该程序的 90％。如果程序员真的写出来了，则认为这个程序具有可理解性，否则这个要重新编写。

② 可靠性

可靠性表明一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。关于可靠性，度量的标准主要有：平均失效间隔时间 MTTF(Mean Time To Failure）、平均修复时间 MTTR(Mean Time To Repair error)、有效性 A(= MTBD／(MTBD+MDT))。
度量可靠性的方法，主要有两类：

§ 根据程序错误统计数字，进行可靠性预测。常用方法是利用一些可靠性模型，根据程序测试时发现并排除的错误数预测平均失效间隔时间 MTTF。

§ 根据程序复杂性，预测软件可靠性。用程序复杂性预测可靠性，前提条件是可靠性与复杂性有关。因此可用复杂性预测出错率。程序复杂性度量标准可用于预测哪些模块最可能发生错误，以及可能出现的错误类型。了解了错误类型及它们在哪里可能出现，就能更快地查出和纠正更多的错误，提高可靠性。

用于可靠性度量的检查表的内容有：程序中对可能出现的没有定义的数学运算是否做了检查？如除以“0”。循环终止和多重转换变址参数的范围，是否在使用前做了测试？下标的范围是否在使用前测试过？是否包括错误恢复和再启动过程？所有数值方法是否足够准确？输入的数据是否检查过？测试结果是否令人满意？大多数执行路径在测试过程中是否都已执行过？对最复杂的模块和最复杂的模块接口，在测试过程中是否集中做过测试？测试是否包括正常的、特殊的和非正常的测试用例？程序测试中除了假设数据外，是否还用了实际数据？为了执行一些常用功能，程序是否使用了程序库？

③ 可测试性

可测试性表明论证程序正确性的容易程度。程序越简单，证明其正确性就越容易。而且设计合用的测试用例，取决于对程序的全面理解。因此，一个可测试的程序应当是可理解的，可靠的，简单的。

用于可测试性度量的检查表的内容有：程序是否模块化？结构是否良好？程序是否可理解？程序是否可靠？程序是否能显示任意的中间结果？程序是否能以清楚的方式描述它的输出？程序是否能及时地按照要求显示所有的输入？程序是否有跟踪及显示逻辑控制流程的能力？程序是否能从检查点再启动？程序是否能显示带说明的错误信息？

对于程序模块，可用程序复杂性来度量可测试性。程序的环路复杂性越大，程序的路径就越多。因此，全面测试程序的难度就越大。

④ 可修改性

可修改性表明程序容易修改的程度。一个可修改的程序应当是可理解的、通用的、灵活的、简单的。其中，通用性是指程序适用于各种功能变化而无需修改。灵活性是指能够容易地对程序进行修改。

测试可修改性的一种定量方法是修改练习。其基本思想是通过做几个简单的修改，来评价修改的难度。设 C 是程序中各个模块的平均复杂性，n 是必须修改的模块数，A 是要修改的模块的平均复杂性。 则修改的难度 D 由下式计算：
D = A / C

对于简单的修改，若 D>1，说明该程序修改困难。A 和 C 可用任何一种度量程序复杂性的方法计算。

用于可修改性度量的检查表的内容有：程序是否模块化？结构是否良好？程序是否可理解？在表达式、数组 / 表的上下界、输入 / 输出设备命名符中是否使用了预定义的文字常数？是否具有可用于支持程序扩充的附加存储空间？是否使用了提供常用功能的标准库函数？程序是否把可能变化的特定功能部分都分离到单独的模块中？程序是否提供了不受个别功能发生预期变化影响的模块接口？是否确定了一个能够当做应急措施的一部分，或者能在小一些的计算机上运行的系统子集？是否允许一个模块只执行一个功能？每一个变量在程序中是否用途

单一？能否在不同的硬件配置上运行？能否以不同的输入／输出方式操作？能否根据资源的可利用情形，以不同的数据结构或不同的算法执行？

⑤ 可移植性

可移植性表明程序转移到一个新的计算环境的可能性的大小。或者它表明程序可以容易地、有效地在各种各样的计算环境中运行的容易程度。

一个可移植的程序应具有结构良好、灵活、不依赖于某一具体计算机或操作系统的性能。

用于可移植性度量的检查表的内容有：是否是用高级的独立于机器的语言来编写程序？ 是否是用广泛使用的标准化的程序设计语言来编写程序，且是否仅使用了这种语言的标准版本和特性？程序中是否使用了标准的普遍使用的库功能和子程序？程序中是否极少使用或根本不使用操作系统的功能？程序中数值计算的精度是否与机器的字长或存储器大小的限制无关？程序在执行之前是否初始化内存？程序在执行之前是否测定当前的输入／输出设备？ 程序是否把与机器相关的语句分离了出来，集中放在了一些单独的程序模块中，并有说明文档？程序是否结构化？并允许在小一些的计算机上分段(覆盖)运行？程序中是否避免了依赖于字母数字或特殊字符的内部位表示，并有说明文件？

⑥ 效率

效率表明一个程序能执行预定功能而又不浪费机器资源的程度。这些机器资源包括内存容量、外存容量、通道容量和执行时间。

用于效率度量的检查表的内容有：程序是否模块化？结构是否良好？程序是否具有高度的区域性(与操作系统的段页处理有关)？是否消除了无用的标号与表达式，以充分发挥编译器优化作用？程序的编译器是否有优化功能？是否把特殊子程序和错误处理子程序都归入了单独的模块中？在编译时是否尽可能多地完成了初始化工作？是否把所有在一个循环内不变的代码都放在了循环外处理？是否以快速的数学运算代替了较慢的数学运算？是否尽可能地使用了整数运算，而不是实数运算？是否在表达式中避免了混合数据类型的使用，消除了不必要的类型转换？程序是否避免了非标准的函数或子程序的调用？在几条分支结构中，是否最有可能为“真“的分支首先得到测试？在复杂的逻辑条件中，是否最有可能为“真“的表达式首先得到测试？

⑦ 可使用性

从用户观点出发，把可使用性定义为程序方便、实用、及易于使用的程度。一个可使用的程序应是易于使用的、能允许用户出错和改变，并尽可能不使用户陷入混乱状态的程序。

用于可使用性度量的检查表的内容有：

ⅰ）程序是否具有自描述性？例如，是否有适应不同读者，并附有实例的程序使用说明？是否有交互形式的 Help 功能？是否一有请求，就能对每一个操作方式作出解释？用户能否很快熟悉程序的使用而无需他人的帮助？是否一有请求，就能很容易地获得当前程序状态信息？

ⅱ）程序是否能始终如一地按照用户的要求运行？例如，程序是否有句法上统一的命令语言和错误信息格式？通过尽量缩小响应时间的差异，程序在相似的条件下，其表现是否也相似？

ⅲ）程序是否让用户对数据处理有一个满意的和适当的控制？例如，程序在交互方式运行时，能否控制中止一项任务，开始或恢复另一项任务？在没有副作用的情形下，程序是否允许处理作废？程序是否允许用户查看后台处理？程序是否有一种易懂的命令语言并允许通过命令组合建立宏指令？程序能否在一旦用户有要求时提供提示信息，帮助用户使用系统？程序能否提供可理解的、非危险性的错误信息？

ⅳ）程序是否容易学会使用？例如，程序是否不需要专门的数据处理知识就能使用？对输入格式、要求和限制的解释是否完整和清楚？在交互系统中，用户输入是否在菜单指示支持下进行？程序是否提供带有纠错提示的错误信息？对交互式系统，是否有“联机“手册？对批处理系统，手册是否容易得到？手册是否是用用户术语写的？

ⅴ）程序是否使用数据管理系统来自动地处理事务性工作和管理格式化、地址分配及存储器组织。

ⅵ）程序是否具有容错性？例如，程序是否容忍典型的输入打字错误？当输入动作需要重复时，程序能否接受简化输入？命令能否简写？程序能否验证输入的数据？

ⅶ）程序是否灵活？例如，程序是否允许以自由形式输入？程序是否可以重复使用而无须对输入值做过多的说明？对用户而言，是否有各种不同的输出选择？程序是否可以针对所选择的运行方式，删除不必要的输入、计算和输出？程序是否允许用户扩充命令语言？程序是否可移植？程序是否允许用户定义自己的功能集和特性集？程序能否以子集形式出现？程序是否允许有经验的用户使用运行较快的版本、简写命令、缺省值等，而让没有经验的用户使用运行较慢的版本，并提供求助命令及监控能力等。

⑧ 其它间接定量度量可维护性的方法

Gilb 提出了与软件维护期间工作量有关的一些数据，可以使用它们间接地对软件的可维护性做出估计。

§ 问题识别的时间；
§ 收集维护工具的时间；

§ 分析、诊断问题的时间；
§ 具体的改错或修改的时间；

§ 局部测试的时间；
§ 集成或回归测试的时间；

§ 修改规格说明的时间；
§ 维护的评审时间；

§ 因管理活动拖延的时间；
§ 恢复时间。

这些数据反映了维护全过程中检错－纠错－验证的周期，即从检测出软件存在的问题开始至修正它们并经回归测试验证这段时间。可以粗略地认为，这个周期越短，维护越容易。

6. 提高可维护性的方法

(1) 建立明确的软件质量目标和优先级

一个可维护的程序应是可理解的、可靠的、可测试的、可修改的、可移植的、效率高的、可使用的。但要实现这所有的目标，需要付出很大的代价，而且也不一定行得通。因为某些质量特性是相互促进的，例如可理解性和可测试性、可理解性和可修改性。但另一些质量特性却是相互抵触的，例如效率和可移植性、效率和可修改性等。因此，尽管可维护性要求每一种质量特性都要得到满足，但它们的相对重要性应随程序的用途及计算环境的不同而不同。所以，应当对程序的质量特性，在提出目标的同时还必须规定它们的优先级。这样有助于提高软件的质量，并对软件生存期的费用产生很大的影响。

(2) 使用提高软件质量的技术和工具

① 模块化

模块化是软件开发过程中提高软件质量，降低成本的有效方法之一。也是提高可维护性的有效的技术。它的优点是如果需要改变某个模块的功能，则只要改变这个模块，对其它模块影响很小；如果需要增加程序的某些功能，则仅需增加完成这些功能的新的模块或模块层；程序的测试与重复测试比较容易；程序错误易于定位和纠正；容易提高程序效率。

② 结构化程序设计

结构化程序设计不仅使得模块结构标准化，而且将模块间的相互作用也标准化了。因而把模块化又向前推进了一步。采用结构化程序设计可以获得良好的程序结构。

③ 使用结构化程序设计技术，提高现有系统的可维护性

§ 采用备用件的方法——当要修改某一个模块时，用一个新的结构良好的模块替换掉整个模块。这种方法要求了解所替换模块的外部（接口）特性，可以不了解其内部工作情况。它有利于减少新的错误，并提供了一个用结构化模块逐步替换掉非结构化模块的机会。

§ 采用自动重建结构和重新格式化的工具(结构更新技术)——这种方法采用如代码评价程序、重定格式程序、结构化工具等自动软件工具，把非结构化代码转换成良好结构代码。

§ 改进现有程序的不完善的文档——改进和补充文档的目的是为了提高程序的可理解性，以提高可维护性。

§ 使用结构化程序设计方法实现新的子系统。

§ 采用结构化小组程序设计的思想和结构文档工具——软件开发过程中，建立主程序员小组，实现严格的组织化结构，强调规范，明确领导以及职能分工，能够改善通信、提高程序生产率；在检查程序质量时，采取有组织分工的结构普查，分工合作，各司其职，能够有效地实施质量检查。同样，在软件维护过程中，维护小组也可以采取与主程序员小组和结构普查类似的方式，以保证程序的质量。

(3) 进行明确的质量保证审查

质量保证审查对于获得和维持软件的质量，是一个很有用的技术。除了保证软件得到适当的质量外，审查还可以用来检测在开发和维护阶段内发生的质量变化。一旦检测出问题来，就可以采取措施来纠正，以控制不断增长的软件维护成本，延长软件系统的有效生命期。
为了保证软件的可维护性，有四种类型的软件审查。

① 在检查点进行复审

保证软件质量的最佳方法是在软件开发的最初阶段就把质量要求考虑进去，并在开发过程每一阶段的终点，设置检查点进行检查。检查的目的是要证实已开发的软件是否符合标准，是否满足规定的质量需求。在不同的检查点，检查的重点不完全相同。如图 7.7 所示。

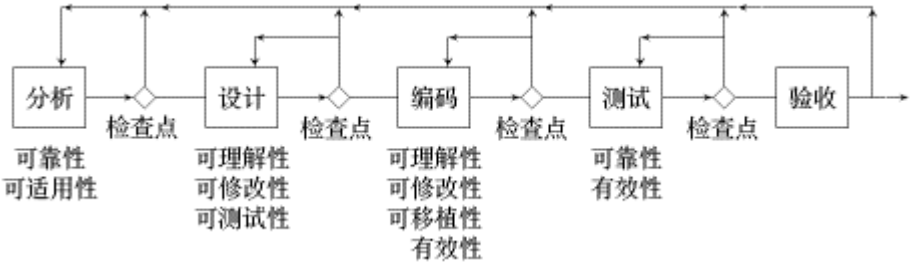


图 7.7 软件开发期间各个检查点的检查重点

② 验收检查

验收检查是一个特殊的检查点的检查，是交付使用前的最后一次检查，是软件投入运行之前保证可维护性的最后机会。它实际上是验收测试的一部分，只不过它是从维护的角度提出验收的条件和标准。

③ 周期性地维护审查

软件在运行期间，为了纠正新发现的错误或缺陷，为了适应计算环境的变化，为了响应用户新的需求，必须进行修改。因此会导致软件质量有变坏的危险，可能产生新的错误，破坏程序概念的完整性。因此，必须像硬件的定期检查一样，每月一次，或二月一次，对软件做周期性的维护审查，以跟踪软件质量的变化。周期性维护审查实际上是开发阶段检查点复查的继续，并且采用的检查方法、检查内容都是相同的。为了便于用户进行运行管理，适时提供维护工具以及有关信息是很重要的。 维护审查的结果可以同以前的维护审查的结果，以及以前的验收检查的结果和检查点检查的结果相比较，任何一种改变都表明在软件质量上或其它类型的问题上可能起了变化。
对于改变的原因应当进行分析。例如，如果使用的是复杂性度量标准，则应当随机地选择少量模块，再次测量其复杂性。如果新的复杂性值大于以前的值，则可能：

§ 是软件可维护性退化的症兆；

§ 预示将来维护该系统需要更多的维护工作量；

§ 表明修改太仓促，没有考虑到要保持系统的完整性；

§ 是软件的文档化工具以及维护人员的专业知识不足所造成的。

反之，若复杂性值减小，则表明软件质量是稳定的。

④ 对软件包进行检查

软件包是一种标准化了的，可为不同单位、不同用户使用的软件。软件包卖主考虑到他的专利权，一般不会提供给用户他的源代码和程序文档。因此，对软件包的维护采取以下方法。使用单位的维护人员首先要仔细分析、研究卖主提供的用户手册、操作手册、培训教程、新版本说明、计算机环境要求书、未来特性表，以及卖方提供的验收测试报告等，在此基础上，深入了解本单位的希望和要求，编制软件包的检验程序。该检验程序检查软件包程序所执行的功能是否与用户的要求和条件相一致。为了建立这个程序，维护人员可以利用卖方提供的验收测试实例，还可以自己重新设计新的测试实例。根据测试结果，检查和验证软件包的参数或控制结构，以完成软件包的维护。

(4) 选择可维护的程序设计语言

程序设计语言的选择，对程序的可维护性影响很大。

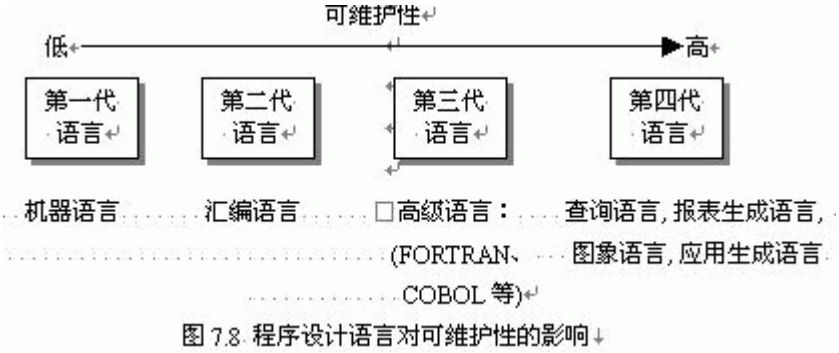


图 7.8 程序设计语言对可维护性的影响

低级语言，即机器语言和汇编语言，很难理解，很难掌握，因此很难维护。高级语言比低级语言容易理解，具有更好的可维护性。但同是高级语言，可理解的难易程度也不一样。
第四代语言，例如查询语言、图形语言、报表生成器、非常高级的语言等，有的是过程化的语言，有的是非过程化的语言。不论是哪种语言，编制出的程序都容易理解和修改，而且，其产生的指令条数可能要比用 COBOL 语言或用 PL / 1 语言编制出的少一个数量级，开发速度快许多倍。有些非过程化的第四代语言，用户不需要指出实现的算法，仅需向编译程序或解释程序提出自己的要求，由编译程序或解释程序自己做出实现用户要求的智能假设，例如自动选择报表格式，选择字符类型和图形显示方式等。总之，从维护角度来看，第四代语言比其它语言更容易维护。

(5) 改进程序的文档

程序文档是对程序总目标、程序各组成部分之间的关系、程序设计策略、程序实现过程的历史数据等的说明和补充。程序文档对提高程序的可理解性有着重要作用。即使是一个十分简单的程序，要想有效地、高效率地维护它，也需要编制文档来解释其目的及任务。而对于程序维护人员来说，要想对程序编制人员的意图重新改造，并对今后变化的可能性进行估计，缺了文档也是不行的。因此，为了维护程序，人们必须阅读和理解文档。

好的文档是建立可维护性的基本条件。它的作用和意义有三点：

§ 文档好的程序比没有文档的程序容易操作。因为它增加了程序的可读性和可使用性。但不正确的文档比根本没有文档要坏得多。

§ 好的文档意味着简洁、风格一致、且易于更新。

§ 程序应当成为其自身的文档。就是说，在程序中应插入注释，以提高程序的可理解，并以移行、空行等明显的视觉组织来突出程序的控制结构。如果程序越

长、越复杂，则它对文档的需要就越迫切。 另外，在软件维护阶段，利用历史文档，可以大大简化维护工作。历史文档有三种：

① 系统开发日志

② 错误记载

③ 系统维护日志

7. 软件再工程

术语“逆向工程”来自硬件。软件公司对竞争对手的硬件产品进行分解，了解竞争对手在设计和制造上的“隐秘”。成功的逆向工程应当通过考察产品的实际样品，导出该产品的一个或多个设计与制造的规格说明。

(1) 逆向工程与再工程的概念

软件的逆向工程是完全类似的。但是，要做逆向工程的程序常常不是竞争对手的，因为要受到法律约束。公司做逆向工程的程序，一般是自己的程序，有些是在多年以前开发出来的。这些程序没有规格说明，对它们的了解很模糊。因此，软件的逆向工程是分析程序，力图在比源代码更高抽象层次上建立程序表示的过程。逆向工程是设计恢复的过程。逆向工程工具可以从已存在程序中抽取数据结构、体系结构和程序设计信息。

再工程，也叫做复壮(修理)或再生。它不仅能从已存在的程序中重新获得设计信息，而且还能使用这些信息来改建或重构现有的系统，以改进它的综合质量。一般软件人员利用再工程重新实现已存在的程序，同时加进新的功能或改善它的性能。

为了执行预防性维护，软件开发组织必须选择在最近的将来可能变更的程序，做好变更它们的准备。逆向工程和再工程可用于执行这种维护任务。

(2) 逆向工程

逆向工程就好象是一个魔术管道。我们把一个非结构化的无文档的源代码或目标代码清单喂入管道，则从管道的另一端出来计算机软件的全部文档。逆向工程可以从源代码或目标代码中提取设计信息，其中抽象的层次、文档的完全性、工具与人的交互程度，以及过程的方法都是重要的因素。如图 7.9 所示。

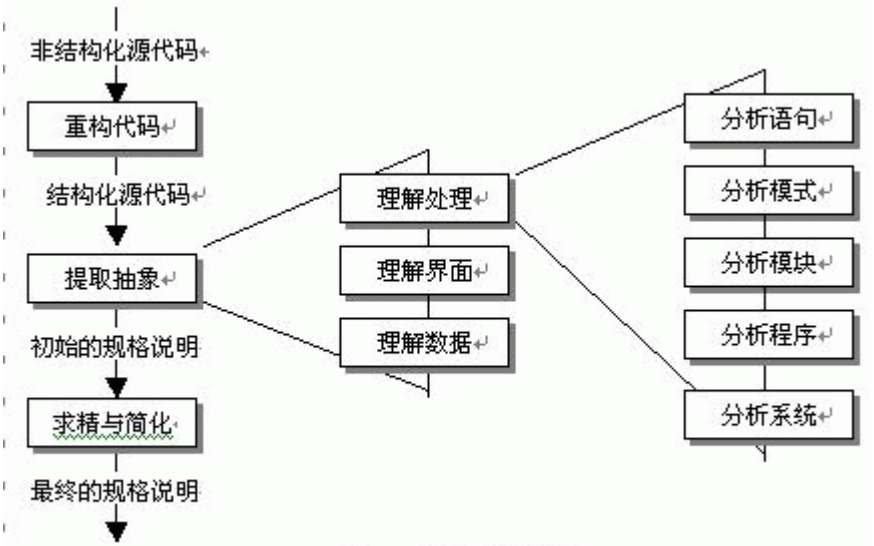


图 7.9 逆向工程过程

逆向工程的抽象层次和用来产生它的工具提交的设计信息是原来设计的赝品，它是从源代码或目标代码中提取出来的。理想情况是抽象层次尽可能地高，也就是说，逆向工程过程应当能够导出过程性设计的表示(最低层抽象)、程序和数据结构信息(低层抽象)、数据和控制流模型(中层抽象)和实体联系模型(高层抽象)。随着抽象层次的增加，可以给软件工程师提供更多的信息，使得理解程序更容易。

逆向工程的文档完全性给出了一个抽象层次所能提供细节的详细程度。在多数情况下，文档完全性随着抽象层次的增加而减少。例如，给出一个源代码清单，可利用它得到比较完全的过程性设计表示；可能还能得到简单的数据流表示；但要得到完全的数据流图则比较困难。

如果逆向工程过程的方向只有一条路，则从源代码或目标代码中提取的所有信息都将提供给软件工程师。他们可以用来进行维护活动。如果方向有两条路，则信息将反馈给再工程工具，以便重新构造或重新生成老的程序。

(3) 软件再工程

再工程组合了逆向工程的分析和设计抽象的特点，具有对程序数据、体系结构和逻辑的重构能力。执行重构可生成一个设计，它产生与原来程序相同的功能，但具有比原来程序更高的质量。

① 为什么要实施软件再工程

§ 再工程可帮助软件机构降低软件演化的风险。当改进原有软件时必须频繁地对软件实施变更，因而降低了软件的可靠性。而软件再工程可以降低变更带来的风险。

§ 再工程可帮助软件机构补偿软件的投资。许多软件机构每年要花费大量的资金用于开发软件。如果采用再工程，而不是扔掉原来的软件，可以部分补偿他们

在软件上的投资。 § 再工程可使得软件易于进一步变更。再工程可使得程序员更容易理解程序，更容易对其开展工作，从而提高维护工作的生产效率。

§ 再工程有着广阔的市场。

§ 再工程是推动自动软件维护发展的动力。

② 软件再工程技术

ⅰ）改进软件

§ 软件重构：软件重构是对软件进行修改，使其易于理解或易于维护。所谓重构，意味着变更源代码的控制结构，它是实现再工程全面自动化的第一步。软件重构的示意图参看图 7.10。

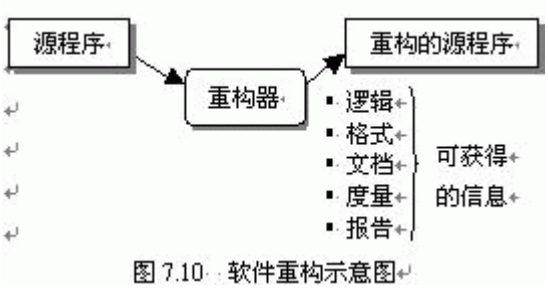


图 7.10 软件重构示意图

§ 文档重写、加注释及文档更新：软件文档重写是要生成更新的校正了的软件信息。重写代码是要将程序代码、其它文档及程序员知识转换成更新了的代码文档。这种文档一般是文本形式的，但可以有图形表示。（包括嵌入的注释、设计和程序规格说明。）用更新文档来实现软件改进是一种早期的软件再工程方法，程序员可以通过嵌入的注释了解程序的功能。文档重写的示意图参见图 7.11。

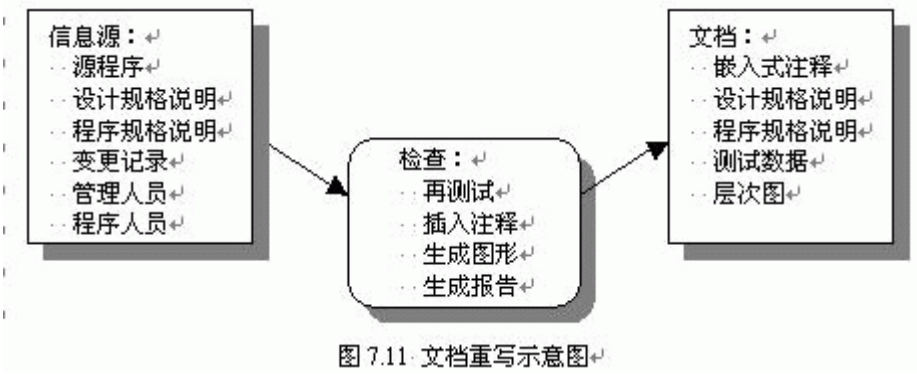


图 7.11 文档重写示意图

§ 复用工程：复用工程的目的是将软件修改成可复用的。通常的做法是：首先寻找软件部件，然后将其改造并放入复用库中。开发新的应用时，可从复用库中选取可复用的构件，实现复用。利用再工程实现复用的过程如图 7.12 所示。

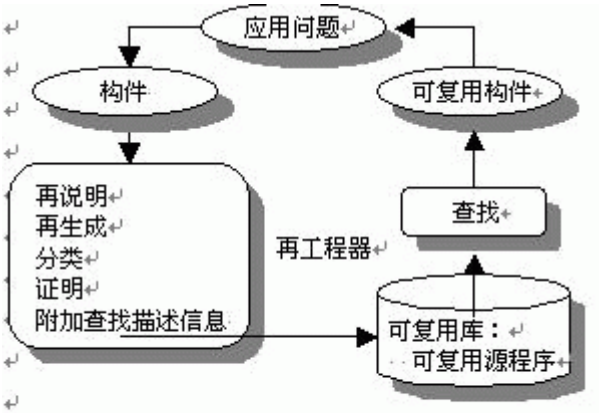


图 7.12 利用再工程实现软件复用的过程

§ 重分模块：重分模块时要变更系统的模块结构，这项工作有赖于对系统构件特性分析和模块耦合性的度量值。

§ 数据再工程：数据再工程是为了改善系统的数据组织，使得数据模式可以辨认和更新。它往往是其它任务(如将数据迁移到其它数据库管理系统)的前期工作。

§ 业务过程再工程：现在的趋势是，使软件去适应业务而不是让业务去适应软件。经验表明，生产率的显著提高有时可能来自在软件帮助下对业务过程所做的自动地重新思考。这种思考可能会导致新的软件设计，新的设计可以成为软件系统再工程、演化的基础。

§ 可维护性分析、业务量分析和经济分析：可维护性分析对于寻找出系统的哪些部分需要再工程十分有用。一般来讲，大多数维护工作往往集中在系统的少数模块。这些部分对于维护成本有着最为强烈的初始冲击。

ii）理解软件

§ 浏览：利用文本编辑器来浏览软件是最早的理解软件的手段。近年，来浏览方法已大有改进，利用超文本，可以在鼠标的帮助下，提供多种视图。如图 7.13 所示。另一种重要的浏览手段是交叉索引。

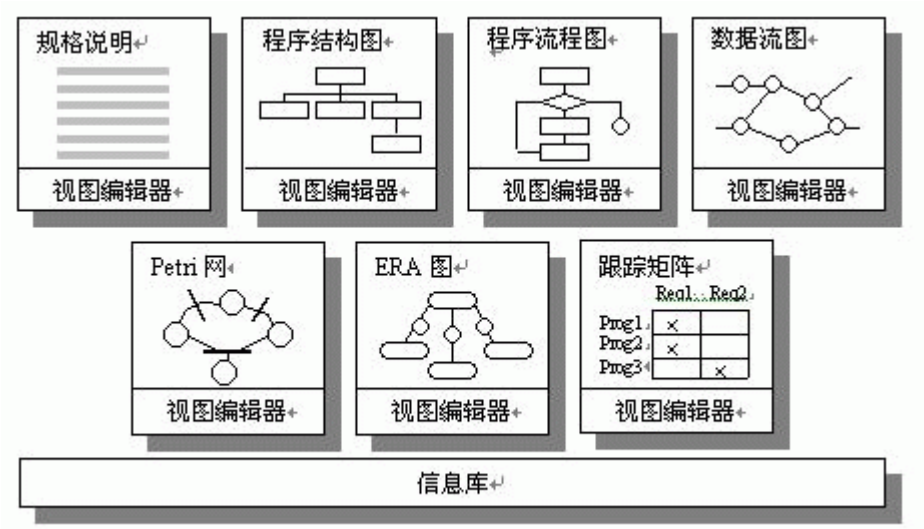


图 7.13 软件视图示例

§ 分析与度量：这也是理解程序特性(如复杂性)的重要方法。软件度量问题已受到软件界的广泛关注。与再工程相关的技术是程序分片、控制流复杂性度量及耦合性度量等。

§ 逆向工程与设计恢复：这两者有相同的含义，都是从另外的途径取得软件信息。这一方法已被人们普遍采用，但用其确定某些设计信息(如设计说明)仍有风险。目前广泛应用的逆向工程是从源程序产生软件设计的结构图或数据流图。

iii）获取、保护和扩充软件的已有知识

§ 程序分解：利用程序分解从程序中找出对象和关系，并将它们存入信息库。而对象和关系一般用于分析、度量以及进一步对信息实施分析和提取。不是直接对源程序实现分解可以节省利用工具进行程序语法分析和生成对象和关系的工作量。

§ 对象恢复：它可以从源程序中取得对象，这可以帮助我们用面向对象的方法来观察以前的一些非面向对象的源程序。面向对象(类、继承、方法、抽象数据类型等)可能是部分的，也可能是全部的。

§ 程序理解：有几种形式。一种是程序员用手工的或自动的方式获得对软件的较好理解，另一种是将有关编程的信息保存起来，再利用这些信息找到编程知识的实例。理解是否正确，需要由软件与编程知识库中信息相匹配的程度决定。

§ 知识库和程序变换：知识库和程序变换是许多再工程技术的基础。变换在程序图上和存于知识库的对象图上进行。为开发新的再工程工具，基于对象的、针对再工程工具的变换结构正在受到广泛关注。

③ 软件再工程的风险

软件再工程是一种软件工程活动，它与任何软件工程项目一样，可能遇到各种风险。软件管理人员必须在工程活动之前有所准备，采取适当对策，防止风险带来的损失。可能出现的风险有以下 6 种。

§ 过程风险：如过高的再工程人工成本；在规定的时间内未达到成本－效益要求；未从经济上规划再工程的投入；对再工程项目的人力投入放任自流；对再工程方案缺少管理的承诺。

§ 人员风险：软件人员对再工程项目意见不一致，影响工作进展；程序员工作低效。

§ 应用风险：再工程项目缺少本应用论域专家的支持；对源程序中体现的业务知识不熟悉；再工程项目的工作完成得不够充分。

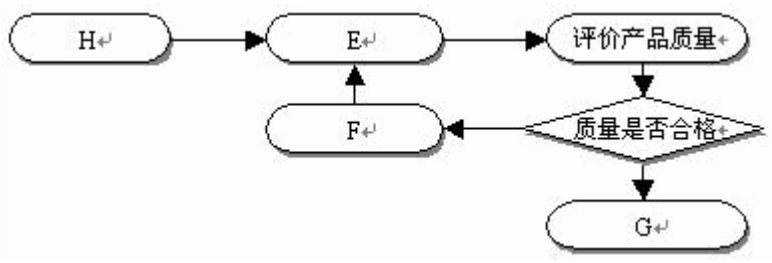
§ 技术风险：恢复的信息是无用的或未被充分利用；大批昂贵的文档被开发出来；逆向工程得到的成果不可共享；采用的再工程方法对再工程目标不适合；缺乏再工程的技术支持。

§ 工具风险：依靠了并非如同广告宣传那样的工具；未经过安装的工具。

§ 策略风险：对整个再工程方案的承诺不成熟；对暂定的目标无长期的打算；对程序、数据和工程过程缺乏全面的观点；无计划地使用再工程工具。

三、例题分析

【例 1】软件产品质量是生产者和用户都十分关心的问题，早期的质量管理只看到产品的质量，近年来质量管理向（ A ）发展，一个重要的基本假设是（ B ）直接影响（ C ）。这一假设最初是根据制造业情况作出的。（ D ）自然就会得到高质量的产品。（ D ）的思想是美国工程师戴明(W. E. Deming)提出来的，戴明等人在质量管理中引入统计质量控制的概念，以降低软件产品缺陷数作为（D）的目标，请选择合适的答案完成下面有关软件质量与过程的关系的示意图。



供选择的答案：

A ~ D. ① 过程的质量 ② 产品的质量 ③ 产品质量的控制

④ 过程质量的控制 ⑤ 过程的改进 ⑥ 产品的改进

⑦ 技术的革新

E ~ H. ① 定义过程 ② 将过程标准化 ③ 开发产品

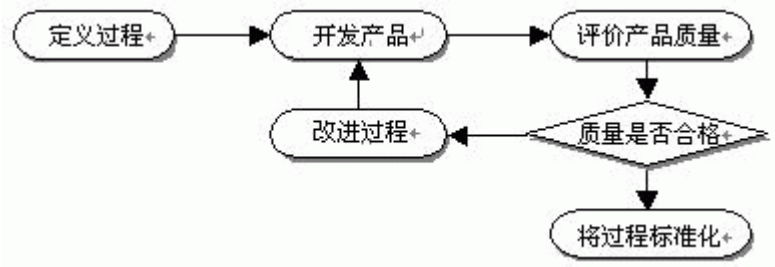
④ 改进过程 ⑤ 质量控制

答案：A. ④, B. ①, C. ②, D. ⑤ E. ③ F. ④ G. ② H. ①

分析：软件产品质量是生产者和用户都十分关心的问题，早期的质量管理只看到产品的质量，实行的是最终产品检验的方法，对质量进行控制。把不合格的产品挑拣出来报废或返工后，得到的产品就是合格的。

近年来质量管理向过程质量的控制方向发展，一个重要的基本假设是开发过程的质量直接影响交付产品的质量。这一假设最初是根据制造业情况作出的。因为在制造业中，产品的质量与生产过程有着十分密切的关系，过程的改进自然就会得到高质量的产品。

过程改进的思想是美国工程师戴明(W. E. Deming)提出来的。二次大战后他到日本工作，在他的指导下，日本产业界一直坚持不断地改进过程，取得了很好的效果。戴明等人在质量管理中引入统计质量控制的概念。这一概念的基础是在产品缺陷数和过程之间建立联系，以降低产品缺陷数作为过程改进的目标。过程改进直到过程成为可以重复的为止，就是说，直到过程的结果成为预期的为止。这时，产品的缺陷数就降下来了，然后再将过程标准化，这时下一步的改进又开始了。产品质量与过程的关系的示意图如下。



【例 2】国际标准化组织和国际电工委员会于 1991 年发布了关于软件质量的标准 ISO／IEC 9126－1991。我国于 1996 年将其等同采用，成为国家标准 GB／T16260－1996《软件产品评价、质量特性及其使用指南》。在此标准中规定了 6 个质量特性及相关的(A)个质量子特性。质量特性包括(B)、(C)、(D)、效率、可维护性和可移植性等。准确性属于(B)，容错性属于(C)，易学习性属于(D)。

供选择的答案：

A. ① 12 ② 16 ③ 21 ④ 22

B ~ D. ① 可靠性 ② 适应性 ③ 可使用性 ④ 安全性

⑤ 一致性 ⑥ 功能性 ⑦ 依从性 ⑧ 互操作性

⑨ 时间特性 ⑩ 资源特性

答案：A. ③, B. ⑥, C. ①, D. ③

分析：在 ISO / IEC 9126－1991 中规定了 6 个质量特性及相关的 21 个质量子特性。如下表所示。这些特性的规定是基于用户的观点的。

功能性		与一组功能及其指定的性质的存在有关的一组属性。功能是指能满足规定或隐含需求的那些功能。
	适合性	与对规定任务能否提供一组功能以及这组功能能否适合有关的软件属性。
	准确性	与能否得到正确的或相符的结果或效果有关的软件属性。
	互操作性	与同其它指定系统进行交互操作的能力有关的软件属性。
	依从性	使软件服从有关的标准、约定、法规及类似规定的软件属性。
	安全性	与避免对程序及数据的非授权故意或意外访问的能力有关的软件属性。
可靠性		与在规定的一段时间内和规定的条件下，软件维持其性能水平有关的能力。
	成熟性	与由软件故障引起失效的频度有关的软件属性。
	容错性	与在软件错误或违反指定接口情况下，维持指定的性能水平的能力有关的软件属性。
	易恢复性	与在故障发生后重新建立其性能水平并恢复直接受影响数据的能力，以及为达此目的所需的时间和有关的软件属性。
可使用性		与为使用所需的努力和由一组规定的或隐含的用户对如此使用所做的评价有关的一组属性。
	易理解性	与用户为理解逻辑概念及其应用性所需努力有关的软件属性。
	易学习性	与用户为学习其应用(例如操作控制、输入、输出)所需努力有关的软件属性。
	易操作性	与用户为进行操作或操作控制所需努力有关的软件属性。
效率		与在规定条件下，软件的性能水平与所用资源量之间的关系有关的一组属性。
	时间特性	与响应和处理时间以及软件执行其功能时的吞吐量有关的软件属性。
	资源特性	与软件执行其功能时所使用的资源量以及使用资源的持续时间有关的软件属性。
可维护性		与进行规定的修改所需努力有关的一组属性。
	易分析性	与为诊断缺陷或失效原因，或为判定待修改的部分所需努力有关的软件属性。
	易变更性	与进行修改、调试或适应环境变化所需努力有关的软件属性。
	稳定性	与修改造成未预料后果的风险有关的软件属性。
	易测试性	为确认经修改软件所需努力有关的软件属性。
可移植性		与软件从一种环境转移到另一环境的能力有关的一组属性。
	适应性	与一软件无需采用有别于为该软件准备的处理和手段就能适应规定的环境有关的软件属性。
	易安装性	与在指定环境下安装软件所需努力有关的软件属性。
	遵循性	使软件服从与可移植性有关的标准或约定的软件属性。
	易替换性	与一软件在该软件环境中用来替代指定的其它软件的可能和努力有关的软件属性。

【例 3】一个软件产品开发完成投入使用后，常常由于各种原因需要对它做适当的变更。在软件的使用过程中，软件原来的(A)可能不再适应用户的要求，需要进行变更；软件的工作环境也可能发生变化，最常见的是配合软件工作的(B)

有变动；还有一种情况是在软件使用过程中发现错误，需要进行修正。通常把软件交付使用后做的变更称为(C)。软件投入使用后的另一项工作是(D)，针对这类软件实施的软件工程活动，主要是对其重新实现，使其具有更好的(E)，包括软件重构、重写文档等。(D)和新的软件开发工作的主要差别在于(H)。我们把常规的软件开发称为(F)，而(G)是从代码开始推导出设计或是规格说明来。

供选择的答案：

A, B. ① 环境 ② 软件 ③ 硬件 ④ 功能和性能 ⑤ 要求

C, D, F, G. ① 逆向工程 ② 正向工程 ③ 软件再工程 ④ 维护 ⑤ 设计

E. ① 可靠性 ② 可维护性 ③ 可移植性 ④ 可修改性

H. ① 使用的工具不同 ② 开发的过程不同

③ 开发的起点不同 ④ 要求不同

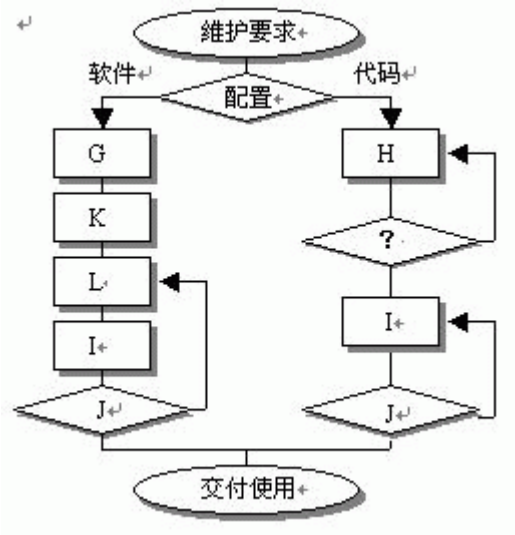
答案：A. ④, B. ③, C. ④, D. ③, E. ②, F ②, G ①, H ③

分析：一个软件产品开发完成投入使用后，常常由于各种原因需要对它做适当的变更。在软件的使用过程中，软件原来的功能和性能可能不再适应用户的要求，需要进行变更；软件的工作环境也可能发生变化，最常见的是配合软件工作的硬件有变动；还有一种情况是在软件使用过程中发现错误，需要进行修正。通常把软件交付使用后做的变更称为维护。软件维护是软件生存期中的一个重要阶段。软件投入使用后的另一项工作是软件再工程，它和上述的软件维护很有关系，但不是一回事，只是近年来才受到重视。

近年来，随着计算机的普及，各产业部门、政府机构等在软件方面的投资迅速增长。几乎所有的部门、机构的活动都采用了计算机。但大量软件的维护不是轻松的事，需要维护的软件越来越多，而维护工作的开销又大得惊人，这就使得许多机构不可能在开发新的软件上再投资来提高自己的工作能力。另一方面，待维护的软件又常常是一些单位业务工作的关键，其中渗入了许多业务知识和工作经验，这些知识和经验除在软件中体现外并没有其它记载。如果这些软件是早期开发的，当时没有采用软件工程技术，程序结构很差，甚至使用的是陈旧的程序设计语言，或者这些软件的文档已经过时，也可能原来就极不完整，当时的开发人员早已不知去向，使用单位没有人能够了解和看懂软件内部的细节，这样的软件不可能对他们做维护工作。针对这类软件实施的软件工程活动，主要是对其重新实现，使其具有更好的可维护性，包括软件重构、重写文档，或是改用新的编程语言，或是将其数据转移到现行的数据库管理系统中，或是在分布式平台上实现等。 软件再工程和新的软件开发工作的主要差别在于开发的起点不同。再工程工作并不是从编写需求规格说明开始，而是将原有的软件作为规格说明。正因为如此，我们把常规的软件开发称为正向工程，而逆向工程是从代码开始推导出设计或是规格说明来。再工程的最终目标仍然是产生新的、更容易维护的新系统。

【例 4】软件维护是软件生存期的最后一个阶段。软件工程学针对维护工作的主要目标是提高(A)，降低(B)。软件的(C)、(D)、(E)是决定软件可维护性的基本因素。软件生存期(F)的工作与软件可维护性有密切的关系。

右图为结构化维护和非结构化维护的对比，请选择合适的答案填入图中。



供选择的答案：

A, B. ① 软件的生产率 ② 文档

③ 软件的可靠性

④ 软件的可维护性 ⑤ 维护的代价

⑥ 维护的效率

C, D, E. ① 可测试性 ② 互操作性 ③ 可理解性 ④ 可修改性 ⑤ 可复用性 ⑥ 可管理性

F. ① 编码阶段 ② 设计阶段 ③ 测试阶段 ④ 每个阶段

G ~ L. ① 复查 ② 计划途径 ③ 重编程序 ④ 评价设计

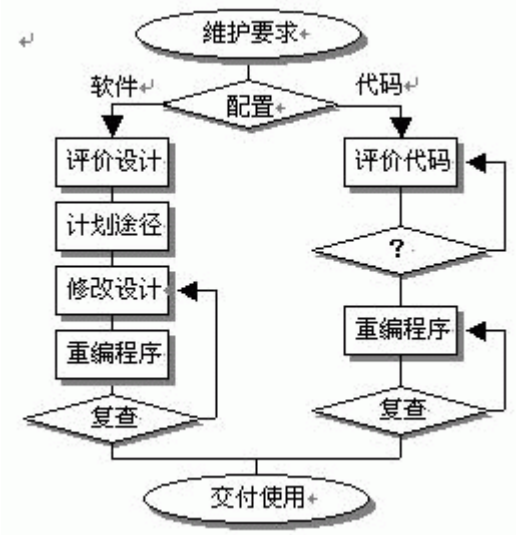
⑤ 评价代码 ⑥ 修改设计

答案：A. ④, B. ⑤, C. ①, D. ③, E. ④, F ④, G ④, H ⑤, I ③, J ①,

K ②, L ⑥。其中，C、D、E 的答案顺序可互换。

分析：软件维护是软件生存期的最后一个阶段，也是持续时间最长、代价最大的一个阶段。软件工程学针对维护工作的主要目标就是提高软件的可维护性，降低维护的代价。软件维护通常包括 4 类维护活动：改正性维护、适应性维护、完善

性维护和预防性维护。软件的可理解性、可修改性和可测试性是决定软件可维护性的基本因素。软件生存期各个阶段的工作都与软件可维护性有密切的关系。在右图中描述了作为维护要求结果可能发生的事件流。



如果软件配置的唯一成分使程序代码，则维护活动从艰难地评价程序代码开始。如果程序内部的文档不足，将使得评价更为困难。诸如软件结构、全局数据结构、系统接口、性能和(或)设计约束等微妙的特征难以搞清，常常会产生误解。最后对程序代码所做变更的后果是难于估量的。因为没有测试方面的文档，所以不可能做回归测试(对修改后的软件进行测试以确认软件的功能没有因修改而退化)。这就是非结构化维护，是我们没有使用良好定义的方法论开发软件的必然结果。因此，必须为此付出代价(大量人力的浪费并要经受挫折)。

如果存在完整的软件配置，则维护工作可以从评价设计文档开始，确定软件重要的结构特征、性能特征以及接口特征，估计要求的改动可能带来的影响，并且计划实施途径。然后首先修改设计并对所做的改动进行仔细的复查。接下来编写相应的源代码程序，使用在测试规格说明中包含的信息进行回归测试。最后，把修改后的软件交付使用。这就是结构化维护。它是在软件开发的早期就应用软件工程方法论的结果。虽然有了完整的软件配置并不能保证维护中一定没有问题，但确实能减少人力的浪费且能提高软件维护的总体质量。

【例 5】软件可维护性是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改，(A)的容易程度。目前广泛使用 7 个特性来衡量软件的可维护性，其中就有(B)、(C)、(D)。其中，(B)和(D)主要在改正性维护中侧重应用，(C)主要在适应性维护和(E)维护中侧重应用。

供选择的答案：

A. ① 维护 ② 扩充与压缩 ③ 调整 ④ 再工程

B ~ D. ① 安全性 ② 可靠性 ③ 完整性 ④ 适应性

⑤ 可理解性 ⑥ 可使用性 ⑦ 一致性 ⑧ 数据无关性

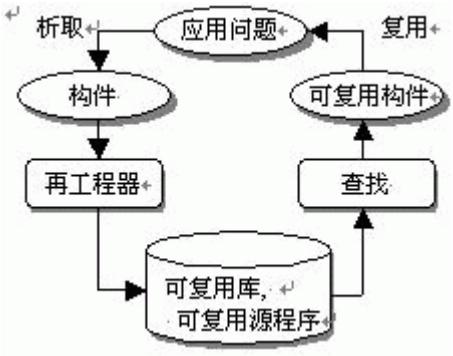
E. ① 预防性 ② 完善性 ③ 改正性 ④ 容错性

答案：A. ②, B. ②, C. ⑥, D. ⑤, E. ②。其中，B、D 的答案顺序可互换。

分析：软件可维护性是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改，扩充或压缩的容易程度。目前广泛使用 7 个特性来衡量软件的可维护性，其中就有可靠性、可理解性和可使用性。其中，可靠性和可理解性主要在改正性维护中侧重应用，可使用性主要在适应性维护和完善性维护中侧重应用。参看下面有关 7 种质量特性在各类维护中侧重点的列表。

	可理解性	可测试性	可修改性	可靠性	可移植性	可使用性	效率
改正性维护	O	O	O	O			
适应性维护			O		O	O	
完善性维护						O	O

【例 6】软件再工程技术主要有(A)、(B)和(C)。分析、度量属于(C)，知识库和变换属于(D)；复用工程属于(E)，其目的是(F)，使其更容易复用。通常是先寻找(G)，然后将其改造，使之能放入(H)中。实际应用可从(H)中选取(I)，以实现复用。



供选择的答案：

A ~ E. ① 改进软件 ② 编码

③ 理解软件 ④ 维护

⑤ 获取、保存及扩充软件知识

F. ① 保存软件 ② 修改软件

③ 设计软件 ④ 开发软件

G ~ I. ① 构件 ② 可复用库

③ 可复用构件 ④ 信息库

答案：A. ①, B. ⑤, C. ③, D. ⑤, E. ①,

F ②, G ①, H ②, I ③ 分析：软件再工程技术包括： i）改进软件，如重构、文档重写、复用工程、重新划分模块、数据再工程、业务过程再工程、可维护分析等。ii）理解软件，如浏览、分析、度量、逆向工程和设计恢复等。iii）获取、保护及扩充软件知识，如分解、逆向工程和设计恢复、对象恢复、程序理解、知识库和变换等。

因此，本例题的解答是：软件再工程技术主要有改进软件、获取、保存及扩充软件知识和理解软件。分析、度量属于理解软件，知识库和变换属于获取、保存及扩充软件知识；复用工程属于改进软件，其目的是修改软件，使其更容易复用。通常是先寻找构件，然后将其改造，使之能放入可复用库中。实际应用可从库中选取可复用的构件，以实现复用。

四、习题

【7-1】对于软件产品来说，有 4 个方面影响着产品的质量，即(A)、(B)、(C)及成本、时间和进度等条件。重视软件过程的质量是近年来质量管理理论和实践的新发展。重视软件过程质量的控制，其部分原因可能是：相对于产品质量的控制来说，过程质量的控制是(D)、(E)、(F)，而产品质量的控制是(G)、(H)、(I)。

供选择的答案：

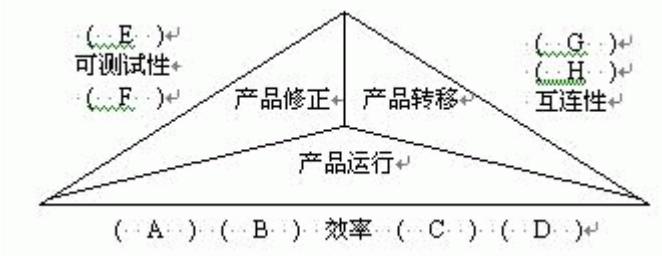
A ~ C. ① 开发时间 ② 开发技术 ③ 过程质量 ④ 风险控制

⑤ 质量控制 ⑥ 人员素质 ⑦ 项目管理 ⑧ 配置管理

D ~ I. ① 主动的 ② 被动的 ③ 整体的 ④ 系统的

⑤ 先期的 ⑥ 事后的 ⑦ 个别的 ⑧ 部分的

【7-2】McCall 提出了表明软件质量的 11 个质量特性。它们是(A)、(B)、(C)、(D)、(E)、(F)、(G)、(H)、效率、可测试性和互连性。我们把这 11 个特性分为 3 组，使其分别隶属于产品修正、产品转移和产品运行等 3 个方面，如图所示。



供选择的答案：

A ~ H. ① 可读性 ② 正确性 ③ 功能性 ④ 完整性 ⑤ 可靠性

⑥ 可移植性 ⑦ 可复用性 ⑧ 灵活性 ⑨ 可维护性 ⑩ 可使用性

【7-3】为什么软件需要维护？维护有哪几种类型？简述它们的维护过程。

【7-4】 在软件维护的实施过程中，为了正确、有效地修改，需要经历以下 3 个步骤：（ A ）、（ B ）、（ C ）。（ A ）是决定维护成败和质量好坏的关键。（ C ）包括（ D ）确认、计算机确认和维护后的（ E ）。

供选择的答案：

A ~ C. ① 修改程序 ② 建立目标程序 ③ 分析和理解程序

④ 重新验证程序 ⑤ 验收程序

D. ① 动态 ② 静态 ③ 人工 ④ 自动

E. ① 验证 ② 验收 ③ 检验 ④ 存档

【7-5】从供选择的答案中选出同下列各叙述关系最密切的字句。

A. 软件从一个计算机系统或环境转移到另一个计算系统或环境的容易程度。

B. 软件在需要它投入使用时能实现其指定的功能的概率。 C. 软件使不同的系统约束条件和用户需求得到满足的容易程度。

D. 在规定的条件下和规定的一段期间内，实现所指定的功能的概率。

E. 尽管有不合法的输入，软件仍能继续正常工作的能力。

供选择的答案：

① 可测试性 ② 可理解性 ③ 可靠性 ④ 可移植性

⑤ 可使用性 ⑥ 兼容性 ⑦ 容错性 ⑧ 可修改性

⑨ 可接近性 ⑩ 一致性

【7-6】改错性维护与“排错”是否是一回事？为什么？

【7-7】从下列叙述中选出 5 条与提高软件的可移植性有关的叙述。

① 把程序中与计算机硬件特性有关的部分集成在一起。

② 选择时间效率和空间效率高的算法。

③ 使用结构化的程序设计方法。

④ 尽量用高级语言编写程序中对效率要求不高的部分。

⑤ 尽可能减少注释。

⑥ 采用表格控制方式。

⑦ 文档资料详尽、正确。

⑧ 在有虚拟存储器的计算机系统上开发软件。

⑨ 减少程序中对文件的读写次数。

⑩ 充分利用宿主计算机的硬件特性。

【7-8】软件可移植性是用来衡量软件的(A)的重要尺度之一。为了提高软件的可移植性，应注意提高软件的(B)。采用(C)有助于提高(B)。为了提高可移植性，还应(D)。使用(E)语言开发的系统软件具有较好的可移植性。

供选择的答案：

A. ① 通用性 ② 效率 ③ 质量 ④ 人机界面

B. ① 使用的方便性 ② 简洁性 ③ 可靠性 ④ 设备独立性

C. ① 优化算法 ② 专用设备 ③ 表格驱动方式 ④ 树型文件目录

D. ① 有完备的文件资料 ② 选择好的宿主计算机 ③ 减少输入输出次数

④ 选择好的操作系统

E. ① COBOL ② APL ③ C ④ SQL

【7-9】下面有关软件维护的叙述有些是不准确的，请将它们列举出来。

供选择的答案：

① 要维护一个软件，必须先理解这个软件。

② 阅读别人写的程序并不困难。

③ 如果文档不齐全也可以维护一个软件。

④ 谁写的软件就得由谁来维护这个软件。

⑤ 设计软件时就应考虑到将来的可修改性。

⑥ 维护软件时一件很吸引人的创造性工作。

⑦ 维护软件就是改正软件中的错误。

⑧ 维护好一个软件是一件很难的事情。

【7-10】什么是程序修改的副作用？程序修改的副作用有哪几种？试举例说明？

【7-11】讨论高级语言对适应性维护的影响。使程序适应新的环境是可能的吗？

【7-12】在软件计划中是否应该把维护费用计划在内？实际情况如何？

【7-13】软件再工程是一类软件工程活动，它能够使我们：ⅰ）增进对软件的理解；ⅱ）准备或直接提高软件自身的（ A ）、（ B ）或演化性。第ⅱ部分旨在改善软件的（C），使得软件更容易为人们服务。纯粹是出于改善性能的代码优化（D）软件再工程。逆向工程属于上述软件再工程的第（ E ）部分。

供选择的答案：

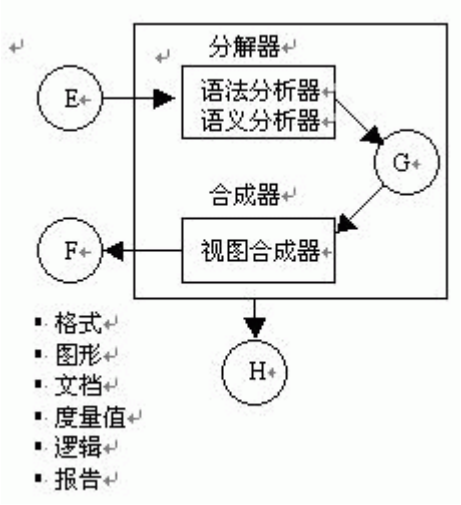
A, B. ① 可靠性 ② 灵活性 ③ 可维护性 ④ 可复用性 ⑤ 可修改性

C. ① 静态质量 ② 动态质量 ③ 性能 ④ 功能

D. ① 属于 ② 不属于

E. ① ⅱ ② ⅰ

【7-14】关于软件再工程的定义有这样两种说法。ⅰ）软件再工程是变更系统(或程序)的（ A ），或是系统(或程序)的（ B ），而不变更其（ C ）的一种工程活动。ⅱ）检查并改进对象系统，按新的模式对系统进行（D），进而实现其新的模式。



多数软件再工程工具可按图示的自动进行再工程的模式工作。请选择合适的答案完成这个图示。 供选择的答案：

A ~ D. ① 外部环境 ② 接口

③ 内部机制 ④ 流程图

⑤ 重构 ⑥ 数据结构

⑦ 功能性 ⑧ 层次性

E ~ H. ① 其它产品 ② 软件工具 ③ 信息库 ④ 软件

⑤ 软件的新视图

【7-15】(A)是软件的一种表达形式，或是有关软件的一种报告。(A)可分为4类：软件的规格说明属于(B)，PDL或DFD属于(C)，规格说明文本的图索引属于(D)，程序和程序段属于(E)。(A)得到工具支持时，就成为(F)，可用其支持加入、变更或浏览信息。

供选择的答案：

A, F. ① 软件说明 ② 概要设计 ③ 软件视图 ④ 信息库

⑤ 视图编辑器 ⑥ 软件设计

B ~ E. ① A类视图 ② 1类视图(非过程性描述和/或元描述)

③ 2类视图(伪过程性描述和/或面向体系结构的描述)

④ 3类视图(纯过程性描述或直接导出的信息)

五、习题解答

【7-1】A. ②, B. ③, C. ⑥, D. ⑤, E. ①, F ④, G ⑥, H ②, I ⑦

其中，A、B、C 答案顺序可互换，D、E、F 答案顺序可互换，G、H、I 答案顺序可互换。

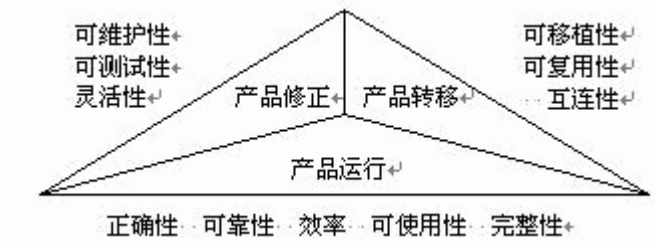
对于软件产品来说，有 4 个方面影响着产品的质量，即开发技术、过程质量、人员素质及成本、时间和进度等条件。这 4 个方面因素对产品质量究竟有多少影响又取决于项目的项目的规模和项目的类型。

重视软件过程的质量是近年来质量管理理论和实践的新发展，但不能把产品质量的控制与过程质量的控制相对立起来。重视软件过程质量的控制，其部分原因可能是，相对于产品质量的控制来说，过程质量的控制是先期的、主动的、系统的，而产品质量的控制是事后的(产品已经生产出来)、被动的(发现了不合格产品只能报废或采取其它补救措施)、个别的(逐个产品的质量检验)。

【7-2】A ②, B ⑤, C ⑩, D ④, E ⑨, F ⑧, G ⑥, H ⑦

其中，A、B、C、D 答案顺序可互换，E、F 答案顺序可互换，G、H 答案顺序可互换。

McCall 质量模型如图所示。用户对于已经投入使用的软件，会特别关注软件运行以后所表现出来的运行特性、修正特性和转移特性。它们直接关系到软件的使用寿命。因此，软件的 11 个质量特性应分为 3 组，分别隶属于这 3 个方面。



其中，属于产品运行特性的有正确性、可靠性、可使用性、完整性和效率。属于产品修正特性的有可维护性、可测试性及灵活性。属于产品转移特性的有可移植性、可复用性和互连性等。

【7-3】在软件开发完成交付用户使用后，为了保证软件在一个相当长的时期能够正常运行，就需要对软件进行维护。

软件维护的类型有 4 种：改正性维护、适应性维护、完善性维护和预防性维护。其中，改正性维护是要改正在特定的使用条件下暴露出来的一些潜在程序错误或设计缺陷；适应性维护是要在软件使用过程中数据环境发生变化或处理环境发生变化时修改软件以适应这种变化；完善性维护是在用户和数据处理人员使用软件

过程中提出改进现有功能，增加新的功能，以及改善总体性能的要求后，修改软件以把这些要求纳入到软件之中。

由这些原因引起的维护活动可以归为以下几类：预防性维护是为了提高软件的可维护性、可靠性等，事先采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试，为以后进一步改进软件打下良好基础。

软件维护的过程如图所示。第一步是先确认维护要求。这需要维护人员与用户反复协商，弄清错误概况以及对业务的影响大小，以及用户希望做什么样的修改，并把这些情况存入故障数据库。然后由维护组织管理员确认维护类型。

对于改正性维护申请，从评价错误的严重性开始。如果存在严重的错误，则必须安排人员，在系统监督员的指导下，进行问题分析，寻找错误发生的原因，进行“救火”性的紧急维护；对于不严重的错误，可根据任务、机时情况、视轻重缓急，进行排队，统一安排时间。对于适应性维护和完善性维护申请，需要先确定每项申请的优先次序。若某项申请的优先级非常高，就可立即开始维护工作，否则，维护申请和其它的开发工作一样，进行排队，统一安排时间。并不是所有的完善性维护申请都必须承担，因为进行完善性维护等于是做二次开发，工作量很大，所以需要根据商业需要、可利用资源的情况、目前和将来软件的发展方向、以及其它的考虑，决定是否承担。

尽管维护申请的类型不同，但都要进行同样的技术工作。这些工作有：修改软件需求说明、修改软件设计、设计评审、对源程序做必要的修改、单元测试、集成测试(回归测试)、确认测试、软件配置评审等。在每次软件维护任务完成后，最好进行一次情况评审，对以下问题做一总结：

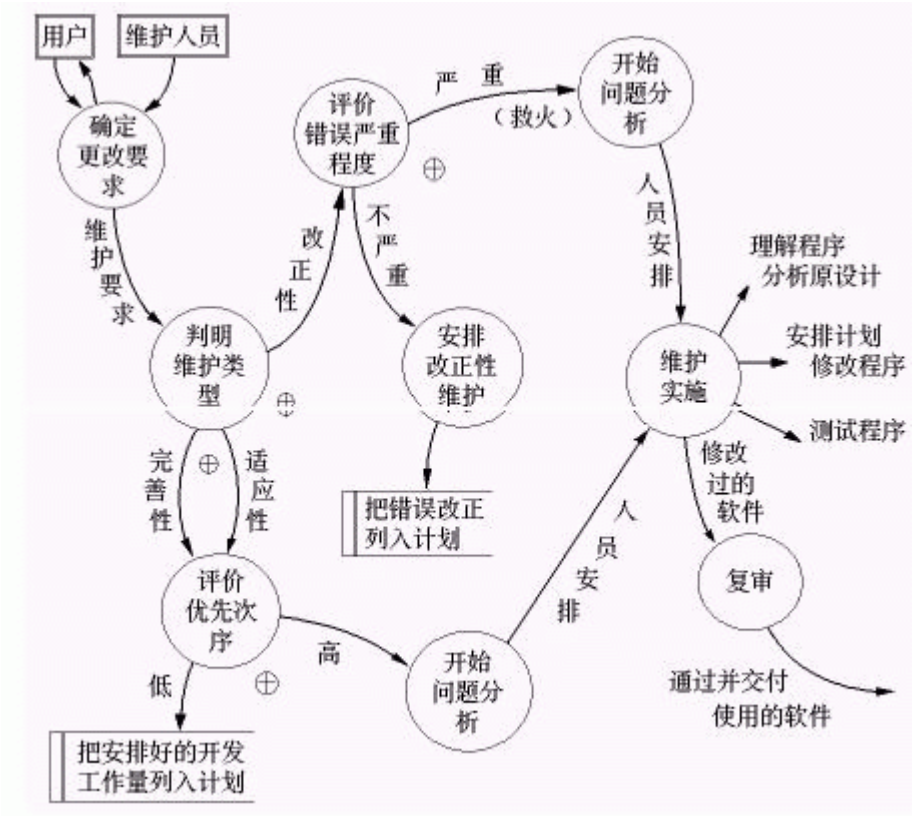
§ 在目前情况下，设计、编码、测试中的哪一方面可以改进?

§ 哪些维护资源应该有但没有?

§ 工作中主要的或次要的障碍是什么?

§ 从维护申请的类型来看是否应当有预防性维护?

情况评审对将来的维护工作如何进行会产生重要的影响，并可为软件机构的有效管理提供重要的反馈信息。



【7-4】A. ③, B. ①, C. ④, D. ②, E. ②

在软件维护的实施过程中，为了正确、有效地修改，需要经历以下 3 个步骤：分析和理解程序、修改程序和重新验证程序。经过分析，全面、准确、迅速地理解程序是决定维护成败和质量好坏的关键。有如下几种方法：分析程序结构图、数据跟踪、控制跟踪及其它方法。在将修改后的程序提交用户之前，需要通过静态确认、计算机确认和维护后的验收，保证修改后的程序的正确性。

【7-5】A. ④, B. ⑤ C. ⑥, D. ③, E. ⑦

论述 A 是指可移植性。可移植性的定义是：将一个软件系统从一个计算机系统或环境移植到另一个计算机系统或环境中运行时所需工作量的大小。

论述 B 是指可使用性。可使用性的定义是：程序方便、实用、及易于使用的程度。用户一有请求，就能对每一个操作方式作出解释，始终如一地按照用户的要求运行。计算其按用户请求实现指定功能的概率，是一种度量准则。

论述 C 是指兼容性。有两类基本的兼容性：向下兼容和交错兼容。向下兼容是软件新版本保留它早期版本的功能的情况；交错兼容是共同存在的两个相关但不同的产品之间的兼容性。软件可以在不同系统约束和不同用户需求下完成指定的工作。 论述 D 是指可靠性。可靠性的定义是：一个程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。

论述 E 是指容错性。容错性的定义是：系统出错(机器临时发生故障或数据输入不合理)时，能以某种预定方式，做出适当处理，得以继续执行和恢复系统的能力。

【7-6】改错性维护与“排错(调试)”不是一个概念。调试是作为测试的后继工作而出现的，是当测试发现软件中的错误后，进一步诊断和改正程序中潜在的错误的活动。而改正性维护是指在软件交付使用后，由于开发时测试的不彻底、不完全，必然会有一部分隐藏的错误被带到运行阶段来，这些隐藏下来的错误在某些特定的使用环境下就会暴露出来。为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用所进行的诊断和改正错误的过程。调试在程序编码阶段、测试阶段、运行和维护阶段都可以发挥作用，它实际上是一种工具或手段。在软件交付运行之后，用户实际充当了测试员的角色，一旦发现软件运行中的错误或缺陷，就会将问题报告通报软件销售商，申请软件维护。其后软件维护人员可以利用调试手段来诊断和改正软件中存在的错误。这时可能涉及的范围不只包括程序，还有文档和数据，不仅可能修改程序代码，而且可能需要修改设计。甚至需求。所以改正性维护是在更大范围中做工作。

【7-7】正确的叙述有 ①、③、④、⑥、⑦。

为了提高软件的可移植性，应当尽可能用高级语言编写源程序代码。对于与硬件或操作系统有关的部分，或对效率要求很高的部分，应当为它们建立专门的模块，将用汇编语言写的程序封装在这些模块中，与程序中其它部分以事先约定的标准方式接口。这样，一旦硬件环境或操作系统环境发生变化，只需修改个别模块即可。

采用表格控制方式，将所有的外部设备接口或与其它系统的接口，包括信息传递、驱动程序入口等都用表格控制，即使将来硬件、相关软件发生的变化，只需修改表格中的登记项，原来的程序一律可以不改。

为了将来修改方便，不致于引入新的错误，相关文档一定要齐全、正确，程序中必须有必要的注释，并使用如结构化程序设计方法这样的良好的程序设计方法来编写程序。至于算法选择，与效率有关，与可移植性无关。其它叙述，如⑧、⑨、⑩，都不利于可移植性。

【7-8】A. ③, B. ④, C. ③, D. ①, E. ③

软件可移植性是用来衡量软件质量的重要尺度之一。为了提高软件的可移植性，应注意提高软件的设备独立性。采用表格驱动的方式有助于提高软件的设备独立性。为了提高可移植性，还应有完备的文档资料。使用 C 语言开发的系统软件具有较好的可移植性。

【7-9】软件维护人员通常不是改软件的开发人员，这给软件维护带来很大的困难。特别是有些软件在开发时没有遵循软件开发的准则，没有开发方法的支持，维护这样的软件就更困难。下面列举一些与软件维护有关的问题。

① 要维护一个软件，首先必须要理解它。而理解一个别人编写的程序通常是很困难的，尤其是对软件配置(指各种相关的文档)不齐全的软件，理解起来就更加困难。

② 需要维护的软件往往缺少合格的文档，或者文档资料不齐全，甚至根本没有文档。在软维护中，合格的文档十分重要，它有助于理解被维护的软件。合格的文档不仅要完整正确地反映开发过程各阶段的工作成果，而且应当容易理解并应与程序源代码一致。而错误的文档会把对软件的理解引入歧途。

③ 在软件维护时，不要指望得到原来开发该软件的人员的帮助。开发人员开发完一个软件后，往往会从事另一软件的开发，甚至已离开原开发单位。即使原来的开发人员还在，也可能时间太久而忘却了实现的细节。。

④ 多数软件在设计时没有考虑到将来的修改，这给软件的修改造成了困难。而且在修改软件时很可能引入新的差错。

⑤ 软件维护通常不是一件吸引人的工作。从事维护工作常使维护人员缺乏成就感，这也严重影响维护工作，从而影响了维护质量的提高。

【7-10】所谓副作用是指因修改软件而造成的错误或其它不希望发生的情况，有以下三种副作用：

① 修改代码的副作用：在使用程序设计语言修改源代码时，都可能引入错误。例如，删除或修改一个子程序、删除或修改一个标号、 删除或修改一个标识符、改变程序代码的时序关系、改变占用存储的大小、改变逻辑运算符、修改文件的打开或关闭、改进程序的执行效率，以及把设计上的改变翻译成代码的改变、为边界条件的逻辑测试做出改变时，都容易引入错误。

② 修改数据的副作用：在修改数据结构时，有可能造成软件设计与数据结构不匹配，因而导致软件出错。数据副作用就是修改软件信息结构导致的结果。例如，在重新定义局部的或全局的常量、 重新定义记录或文件的格式、增大或减小一个数组或高层数据结构的大小、修改全局或公共数据、重新初始化控制标志或指针、重新排列输入／输出或子程序的参数时，容易导致设计与数据不相容的错误。

③ 文档的副作用：对数据流、软件结构、 模块逻辑或任何其它有关特性进行修改时，必须对相关技术文档进行相应修改。否则会导致文档与程序功能不匹配，缺省条件改变，新错误信息不正确等错误。使得软件文档不能反映软件的当前状态。如果对可执行软件的修改不反映在文档里，就会产生文档的副作用。例如，对交互输入的顺序或格式进行修改，如果没有正确地记入文档中，就可能引起重大的问题。过时的文档内容、索引和文本可能造成冲突，引起用户的失败和不满。

【7-11】所谓适应性维护，是指修改软件以适应外部环境(新的硬、软件配置)或数据环境(数据库、数据格式、数据输入／输出方式、数据存储介质)发生的变化。

使用高级语言，如 FORTRAN，C 等，它们以接近自然语言的方式编写程序。这样的程序可理解性强，结构化、模块化都很好，除了与硬件、相关软件(如操作系统)和数据库管理系统有关的部分外，都可以用它们编写程序。但每一种类型的计算机，即使提供同一种高级语言，也有可能渗入它自己特有的功能。为作到可移植，必须避开这些特殊的功能，使用语言的标准文本来编写程序。

当前。许多软件开发商以一些高级语言为核心，开发了众多功能强大的软件编程环境,如 Delphi(以 Pascal 为编程语言),Vasual C++(以 C++为编程语言),Vasual Basic(以 Basic 为编程语言)，Powerbuilder(以 C++为编程语言)等。在这些软件中提供了大量的控件或类库，可以帮助程序员以最小的工作量实现多种功能，还可以与多种数据库接口。所有这些，对于适应性维护，都有有利的影响。但事情也不是绝对的。在 MS Windows*.* 或 NT 环境下开发的软件不一定能够直接转移到 Linux 环境下。虽然许多人在这方面正在努力。

国外一些计算机厂商在硬、软件结合的环境下综合考虑软件跨平台使用的问题。例如，Oracle 数据库管理系统在几乎所有的硬件平台上，微软和苹果的 Windows 环境下都可以安装，这样使用 Oracle SQL 编写的程序在多种硬件平台上，多种数据库和操作系统的环境下，不用修改程序，或修改少量程序，就可以直接运行。

【7-12】针对不同的工作目标，软件计划的可以有以下多种类型：
① 项目实施计划（或称为软件开发计划）——这是软件开发的综合性计划，通常应包括任务、进度、人力、环境、资源、组织等多个方面。
② 质量保证计划——把软件开发的质量要求具体规定为每个开发阶段可以检查的质量保证活动。
③ 软件测试计划——规定测试活动的任务、测试方法、进度、资源、人员职责等。
④ 文档编制计划——规定所开发项目应编制的文档种类、内容、进度、人员职责等。
⑤ 用户培训计划——规定对用户进行培训的目标、要求、进度、人员职责等。
⑥ 综合支持计划——规定软件开发过程中所需要的支持，以及如何获取和利用这些支持。
⑦ 软件分发计划——软件开发项目完成后，如何提供给用户。

并未专门考虑软件的维护费用问题。但实际上，为了提高软件的可维护性，在软件开发的具体操作时，必须考虑将来的维护，采取必要的措施，以降低将来维护的费用。

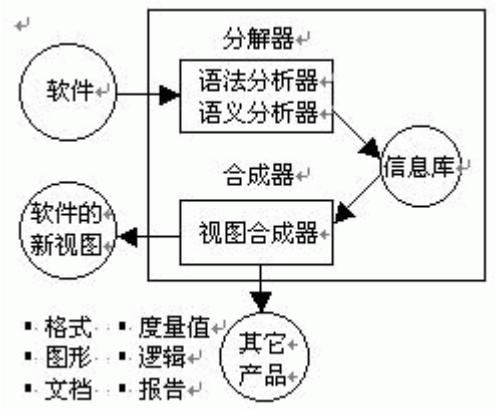
【7-13】A ③, B ④, C ①, D ②, E ②。其中，A、B 答案的顺序可互换。

软件再工程是一类软件工程活动，它能够使我们： ｉ）增进对软件的理解； ii）准备或直接提高软件自身的可维护性、可复用性或演化性。上述软件再工程的第 ii 部分旨在改善软件的静态质量，使得软件更容易为人们服务。但是如果修改的目的并不是为了提高可维护性、可复用性或演化性或是帮人们更好地理解，那么应当把这种修改排除在软件再工程之外。如纯粹是出于改善性能的代码优化或对其重构，都不能算做软件再工程。逆向工程是属于上述软件再工程的第 i 部分。

因为逆向工程所得到的软件表达信息(如源程序)能够帮助人们理解该软件，或是便于对它做进一步的处理。

【7-14】A ③, B ⑥, C ⑦, D ⑤, E ④, F ⑤, G ③, H ①。其中，A、B 的答案顺序可互换；

关于软件再工程的定义有这样两种说法。



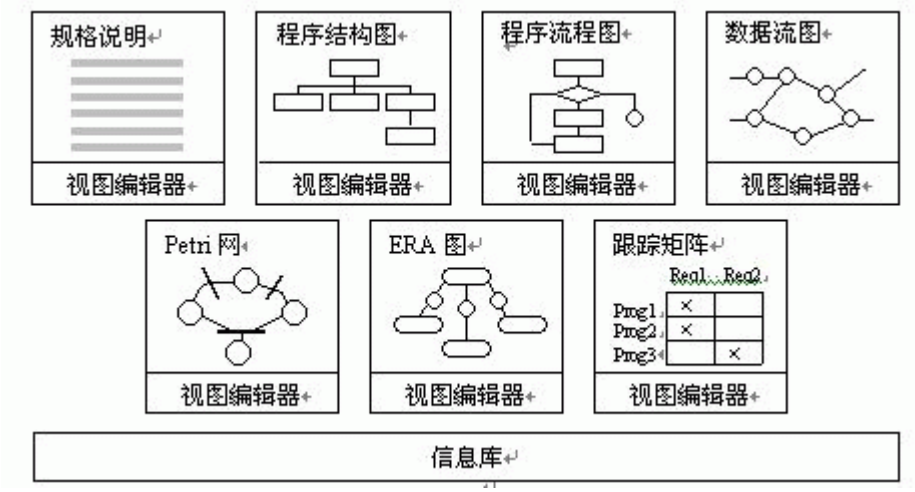
ⅰ)软件再工程是变更系统(或程序)的内部机制，或是系统(或程序)的数据结构，而不变更其功能性的一种工程活动。

ⅱ）检查并改进对象系统，按新的模式对系统进行重构，进而实现其新的模式。

多数软件再工程工具可按图示的自动进行再工程的模式工作。图中大的矩形框表示某个再工程工具。在分解器中进行语法分析和语义分析之后，将相关信息送到信息库。信息库中的信息经过合成器，可以在软件维护中生成各种视图和其它产品，对于进一步分析，自动进行维护以及进行相关的研究都很有价值。

【7-15】A. ③, B. ②, C. ③, D. ①, E. ④, F ⑤

软件视图的示例如下图所示。



有规格说明、数据流图(DFD)、源程序、度量值、根据静态源程序导出的报告、以及表明软件性能的测试数据等。图中的软件视图包含了视图信息。在视图得到视图编辑器的支持时，可以使用这种工具加入、变更和浏览视图的信息。

视图分为 4 类：

① 1 类视图：非过程性描述和／或元描述。例如，软件的规格说明和数据库的概念模式即属于此类。

② 2 类视图：伪过程性描述和／或面向体系结构的描述。例如，软件设计、PDL 描述和软件的体系结构(如模块调用层次图、数据流图 DFD)即属于此类。

③ 3 类视图：纯过程性描述或直接导出的信息。例如，源程序、程序段、数据、源程序中的数据定义、由视图分解得到的对象、关系和语法树均属于此类。

④ A 类视图：伴随以上各类的分析视图。

§ A1 类视图是对 1 类视图的分析信息。例如，规格说明文本的图索引。

§ A2 类视图是有关 2 类视图的分析信息，如源模块的耦合度。

§ A3 类视图是有关 3 类视图的分析视图，如源程序中的模块数。

下面将有关视图的概念解释一下。

① 信息库：信息库是有关软件的信息仓库。通常，把有关软件的信息存入信息库，有 3 种方法：

§ 将软件分解成为对象和关系；

§ 逐步得到对象和关系，借助工具将它们送入信息库；

§ 从其它信息库引入信息。

① 分解：把视图变换成信息库中存放的对象和关系。例如，编译器将程序分解为抽象的语法树表示。

② 合成：根据信息库中的信息生成视图信息。合成器(工具或完成合成工作的人员)在信息库中寻找有关的对象和关系，组合成视图信息，然后根据要求将视图格式化，以显示视图信息。例如，编辑器的后端常常借助于扫视语法树或其它类似成分产生程序代码。

③ 变换：变换是再工程的核心活动。再工程可在同一类视图或前一类视图上把一个软件视图的信息变换成为另一个视图的信息。例如，再工程变换可以是把源代码(属 3 类视图)变换成结构化源代码(3 类视图)、更新的设计(2 类视图)、修正的规格说明(1 类视图)，或是计算出的静态度量值(A3 类视图)。再工程变换通

常按某些准则来改进视图中的信息。软件重构则是变换源代码结构(语法和语义)的变换。

变换也是逆向工程和设计恢复的基础。逆向工程与再工程十分相似，只是两者的原始视图和目标视图不同，逆向工程的目标视图一般在前一类视图。例如，把源代码变换成程序结构图，既可以看成是再工程，也可以看成是逆向工程。但把源代码变换成重构的源代码则不是逆向工程的工作，而是再工程的任务了。 正向工程也是一种变换。它是从较前类的软件视图变换到较后类的软件视图的活动。例如，从数据流图(DFD)生成源程序代码，就是正向工程的活动。

第八章 软件复用

一、复习要求

1. 了解软件复用的概念、过程和意义。

2. 了解软件复用的类型、复用的目标和内容。

3. 了解针对复用的过程模型和论域工程。

4. 了解可复用构件的构造原则和质量标准。

5. 了解可复用构件库的组织，包括可复用构件的 3C 模型。

6. 了解软件的分层式体系结构。

7. 了解软件构件的复用过程(包括构件的检索与提取、理解与评价和修改)

8. 了解面向对象的软件复用技术

9. 了解支持软件复用的 CASE 工具需求

二、内容提要

1. 软件复用的概念、过程和意义

(1) 软件复用的概念 对建立软件目标系统而言，所谓复用(Reuse)，就是利用某些早先开发的对建立新软件系统有用的软件元素来生产新系统。它是一项活动，而不是一个对象。这里所说的软件元素可以包括需求规格说明、设计过程、设计规格说明、程序代码、测试用例、度量等等。对于新的软件开发项目而言，它们或者是构成新软件系统的构件，或者是在软件开发过程中发挥某种作用。通常将这些软件元素称为可复用构件。

早在 50 年代用机器语言编写程序的时期，计算正弦、余弦、对数等的标准子程序包就开创了复用的先河。到 70 年代的早期，在语言、数据结构、操作系统、程序变换等方面各种技术的发展，都与代码的复用有关。当时，复用的应用领域有限，软件的复用问题没有提到很重要的位置上来讨论。70 年代的中期和后期开始，为了缓解软件危机，许多人寄希望于软件复用技术，因为在提高软件生产率与质量，以及减少软件开发时间和成本的努力中软件复用已经成了关键因素。当前，美、日一些大公司的资料表明，软件复用率最高可望达到 90%，而且软件复用使得企业在及时满足市场、软件质量、软件开发费用和维护费用等方面得到显著的改进。

为了保证软件的复用，需要注意解决的问题主要有：

§ 复用必须以对被复用对象的理解为基础；

§ 如何表达对软件构件的需求；

§ 如何寻找有关的软件构件；

§ 如何识别是否满足要求；

§ 如何调整或修改构件使之满足要求；

§ 如何组装到新系统中；

§ 如何保证结果的正确性；

§ 如何进行效益的定量估算。只有当复用能带来效益时人们才会考虑复用。

为了能够在软件开发过程中复用现有的软件构件，必须在此之前不断地进行可复用构件的积累，并且将它们组织成可复用构件库。因此，软件复用不但要讨论如何检索所需的软件构件以及如何对它们进行必要的修剪，而且还要解决如何选取可复用构件、如何组织可复用构件库等问题。

(2) 软件复用的过程和意义

软件复用可以区分为横向复用和纵向复用。横向复用是复用不同应用论域中的软件元素，例如数据结构、排序算法、人机界面构件等。标准函数库是一种典型的原始的横向复用机制。纵向复用是在一类具有较多公共性的应用论域之间复用软件构件。由于在两个截然不同的应用论域之间进行软件复用潜力不大，所以纵向复用受到广泛关注。

纵向复用活动的主要关键点在于论域分析：根据应用论域的特征和相似性，预测软件构件的可复用性。一旦根据论域分析确认了软件构件的可复用价值，即可进行软件构件的开发，并对具有可复用价值的软件构件做一般化处理，使它们能够适应新的类似的应用论域。然后将软件构件和它们的文档存入可复用构件库，成为可供未来开发项目使用的可复用资源。这些步骤构成软件构件的构造活动。这

是一个不断积累、不断完善软件构件的渐进过程。

随着可复用构件的不断丰富，可复用构件库的规模会不断扩大，这样，必须考虑如何组织库的结构以保持较高的检索效率。可供选择的软件构件从库中被检索出来之后，用户还必须理解它的功能或行为，以判定它是否真正适合于当前的应用。必要时，可考虑对某个与期望的功能或行为匹配程度最好的可复用构件进行少量修改，甚至可将修改后的构件再加到可复用构件库中。上述的软件复用的方法如图 8.1 所示。

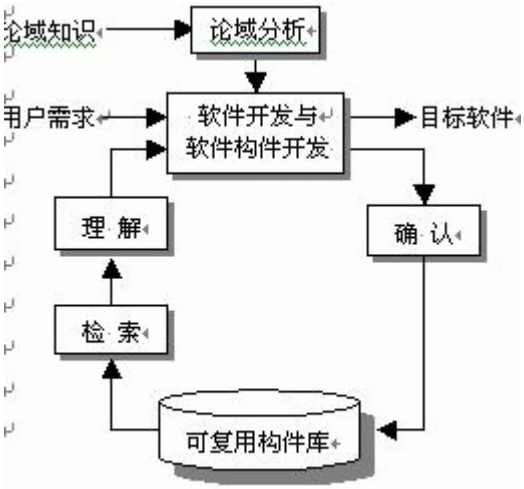


图 8.1 面向复用的软件开发

软件复用的过程可借助计算机的帮助。支持软件复用的 CASE 工具的主要工作是：用某种组织形式实现可复用构件的存储，提供友好的人机界面，帮助用户浏览、检索和修改可复用构件库，对用户感兴趣的软件构件进行解释。事实上，几乎所有的软件复用活动都是在 CASE 工具的帮助下进行的。

使用复用技术可以减少软件开发活动中大量的重复性工作，这样就能够提高软件生产效率，降低开发成本，缩短开发周期。同时。由于可复用构件大都经过严格的质量验证，并在实际运行环境中得到检验，因此，复用软件构件有助于改善软件质量。此外，大量使用可复用构件，软件的灵活性和标准化程度也可望得到提高。

2. 软件复用的类型

软件复用的范围不仅涉及源程序代码，Caper Jones 定义了十种可能复用的软件要素：

① 项目计划：软件项目计划的基本结构和许多内容(如 SQA 计划)都可以跨项目复用。这样可减少制定计划的时间，也可降低与建立进度表、风险分析及其他特征相关的不确定性。

② 成本估计：由于不同项目中常包含类似的功能，所以有可能在极少修改或不修改的情况下，复用对该功能的成本估计。

③ 体系结构：即便应用论域千差万别，但程序和数据体系结构大同小异。因此，可以创建一组类属的体系结构模板(例如事务处理结构)，将这些模板作为可复用的涉及框架。

④ 需求模型和规格说明：类和对象的模型和规格说明显然可以复用。此外，用传统软件工程方法开发的分析模型(如数据流图)也可以复用。

⑤ 设计：用传统方法开发的体系结构、数据、接口和过程化设计都可以复用。另外，复用系统和对象的设计是屡见不鲜的。

⑥ 源程序代码：验证过的程序构件(用兼容的程序设计语言书写)是可以拿来复用的。

⑦ 用户文档和技术文档：即便特定的应用不同，但也经常有可能复用用户文档和技术文档中的大部分内容。

⑧ 用户界面：这可能是最广泛被复用的软件元素。如经常复用 GUI 的软件构件。因为它可占到一个应用的 60%的代码量，所以复用的效果最明显。

⑨ 数据结构：经常被复用的数据结构包括：内部表、列表和记录结构，以及文件和完整的数据库。

⑩ 测试用例：只要将设计或代码构件定义成可复用构件，相关的测试用例就应当成为这些构件的“从属品”。

表 8.1 给出的数据(来自军方的一些系统项目)表明 1 美元投资在 4 年后的回报。Jones 对复用的整体影响给出高度评价：“复用所有 10 种软件要素的总计值可以产生可能是任意已知的软件技术种的最好回报。”

表 8.1 软件复用在 4 年后的回报价值

可复用软件要素	4 年后的回报	可复用软件要素	4 年后的回报
项目计划	$2.00	源程序代码	$6.00
成本估计	$3.00	用户文档和技术文档	$1.50
体系结构	$1.50	用户界面	$1.00
需求模型和规格说明	$3.00	数据结构	$3.50
设计	$5.00	测试用例	$3.50

需要注意的是，复用还可以扩展到以上所讨论的可交付的软件要素之外，它还包含了软件工程过程中的元素。特定的分析建模方法、检查技术、测试用例设计技术、质量保证过程，以及很多其它软件工程实践都可以被“复用”。例如，如果某项目组有效地应用了某种软件工程方法，那么在开发其它项目时可以复用该方法。为帮助潜在的该方法的使用者选择这个方法，应定义一组功能描述，以帮助人们作出适当的决策。

3. 论域工程

(1) 针对复用的过程模型

针对复用的过程模型如图 8.2 所示。这种过程模型强调并行的工作方式。以这种方式，论域工程和基于构件的应用开发同时进行。

论域工程创建应用论域的模型，这个模型将成为软件工程流中分析用户需求的基础。软件体系结构(及相应的结构点)为应用的设计提供了输入。最后，当可复用构件构造好并放入可复用构件库中(在论域工程中进行)之后，它们就可以在软件构造活动中供软件开发人员使用。

(2) 论域工程

论域工程的目的是标识、构造、分类和传播一组软件要素。从软件工程的观点来看，论域就是向应用软件提供应用需求的问题和背景知识。论域工程的主要任务就是针对单个或一族相似的论域，以软件复用为目标，探寻并挖掘论域或论域族中能够为多个应用软件系统共用的软件要素，并对它们进行结构化组织，放入可复用构件库，以备复用。

论域工程包括三个主要的活动：分析、构造和传播。

① 论域分析过程

论域分析与常规的需求分析活动有相似之处，它们都是软件开发中直接基于应用论域的开发活动，并且都要完成程度不同的论域建模任务。但是论域分析必须具有比需求分析更为广阔的视角：不仅要服务于当前的应用，而且要从论域的历史项目中发现这些项目之间的共同点和差异点，并放眼于同类或相似应用论域的未来软件项目。

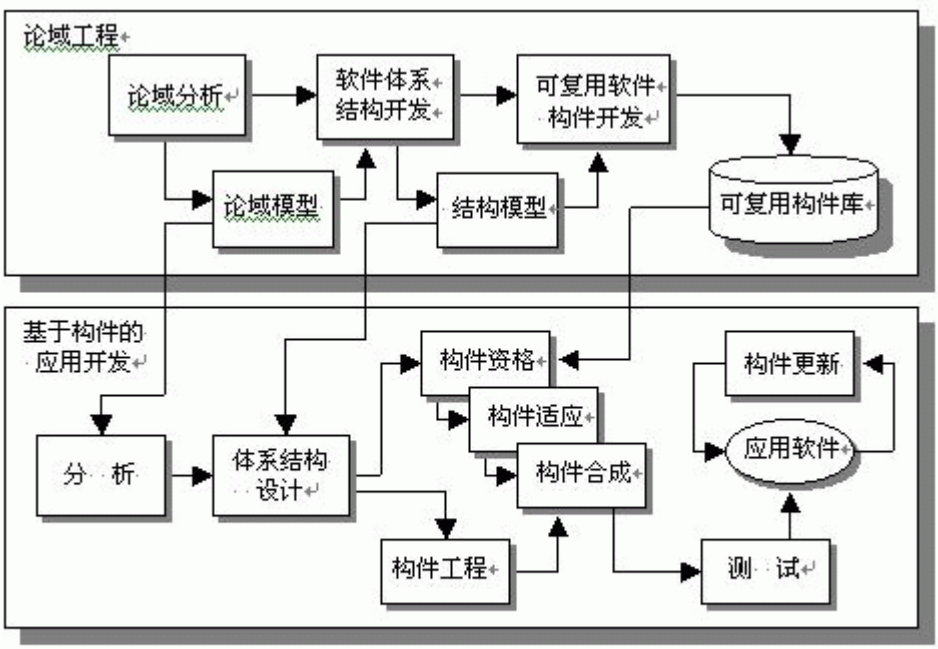


图 8.2 针对复用的过程模型

论域分析的过程如下：

§ 定义将要研究的论域；

§ 分类从论域中抽取出来的项；

§ 收集论域中应用的有代表性的样本；

§ 分析样本中的每一个应用；

§ 开发对象的分析模型。

应当注意的是，论域分析适合于任何一种软件工程范型。Prieto-Diaz 扩展了上面论域分析的第 2 个步骤，建议了一个有 8 个步骤的标识和分类可复用软件构件的方法：

§ 选择特定的功能或对象；

§ 抽象功能或对象；

§ 定义分类方法；

§ 标识公共特征；

§ 标识特定的关系；

§ 抽象关系；

§ 导出功能模型；

§ 定义论域语言。

论域语言提供了在论域中进行应用的规格说明和构造的手段。下面给出一个检查表，以确定哪些软件要素可以成为可复用的构件。

§ 构件的功能在未来的实现工作中需要吗？

§ 在论域中构件功能的公共性如何？

§ 在论域中存在构件功能的副本吗？

§ 构件是否依赖于硬件？

§ 在不同的实现之间硬件是否保持不变？

§ 硬件细节能否移到另一个构件中吗？

§ 设计为下一步的实现进行过足够的优化吗？

§ 我们能够把一个不可复用的构件参数化以使其成为可复用的构件吗？

§ 构件是否可以仅仅经过少量修改就能够在很多实现中复用吗？ § 通过修改实现复用是可行的吗？

§ 一个不可复用的构件能否通过分解以产生一组可复用的构件吗？

§ 针对复用的构件分解有效到什么程度？

② 论域特征

有时很难确定一个潜在可复用的软件构件在某种特定的情况下是否确实可以使用。为了解决这个难题，有必要定义一组可以为论域中所有软件共享的论域特征。论域特征定义了存在于论域中的所有产品的类属属性。例如，类属属性可能包括有：安全性 / 可靠性的重要性、程序设计语言、处理中的并发性，以及其它许多内容。

若设某一可复用软件构件 p 的论域特征的集合为 {Dp}，集合中每一项 Dpi 表示某一特定的论域特征。赋予 Dpi 的值表示该论域特征对软件的相关程度，共分为 5 个等级：

§ 与新软件是否可复用没有相关性。

§ 仅在不寻常的情况下相关。

§ 相关。此时可以修改软件构件以使其可以被复用。

§ 显著相关，且若新软件不具有此特征则复用是低效的。但复用仍有可能。

§ 显著相关，且若新软件不具有此特征则复用是无效的。此时不推荐复用。

如果想要在某应用论域中构造一个新软件 w，可为它导出一组论域特征，然后在 Dwi 与 Dpi 之间进行比较，以决定既存的软件构件 p 是否可以有效地在应用 w 中复用。

表 8.2 列出典型的对软件复用可能有影响的论域特征，为了有效地复用软件构件，必须考虑这些论域特征。

表 8.2 影响复用的论域特征

产品	过程	人员
需求稳定性	过程模型	动机
并发软件	过程符合性	教育
内存限制	项目环境	经验／培训
应用大小	进度限制	▪ 应用论域
用户界面复杂性	预算限制	▪ 过程
程序设计语言	生产效率	▪ 平台
安全性／可靠性		▪ 语言
寿命需求		开发队伍
产品质量		生产效率
产品可靠性		

③ 结构建模和结构点

结构建模是一种基于模式的论域工程方法。使用该方法的前提是：每个应用论域有可在应用间重复的模式(功能的、数据的、行为的)，它们具有复用的可能。那么，什么是结构建模呢？Pollak 和 Rissman 这样描述：结构模型由数量不多的一些结构元素构成，用这些结构元素可明确地表示交互的模式。使用结构模型的系统体系结构可以通过由这些模型元素组成的多声部合唱来特征化。这样，许多体系结构单元都可以通过这些元素中的一些简单的交互模式来描述。 每一个应用论域都可以一个结构模型来特征化。如飞行器电子设备系统虽然在细节上有很大的不同，但在这个论域中的所有的现行软件都有相同的结构模型。因此，结构模型是一种能够而且也应该跨越该论域所有应用的可复用的体系结构要素。

结构点是结构模型中的各个不同的构成成分，用于描述系统体系结构中的模式。它有 3 个特征：

§ 结构点是一个抽象，它应当有有限数量的实例。若用面向对象的术语来陈述，它的类层次的规模应比较小。此外，该抽象应当在论域的各个应用中重复出现，否则，验证、文档化和传播结构点的代价可能是不划算的。 § 使用结构点的管理规则应是容易理解的，使用结构点的界面应比较简单。

§ 结构点的实现应做到信息隐蔽，将所有在结构点内部包含的复杂性封装(或隔离)在结构点内部。这样，可以减少整个系统的感知复杂性。

4. 可复用构件的构造原则和质量标准

论域分析的结果为可复用构件的选取提供了指导原则。一旦确定了某些软件元素应当称为可复用构件并要加入到可复用构件库中，软件人员就必须实际构造它们。由于软件构件的生存期将跨越开发项目，甚至应用论域，因此，可复用构件必须更为通用、更容易组装到新的软件系统中、再新的运行环境下能表现出更好的健壮性。 代码极复用时最简单，也是用得最多的。但可复用软件构件不仅包括代码级构件，它还应涵盖论域知识、需求分析文档、设计文档、测试方案、测试用例等。

(1) 需求和设计级的软件构件

需求级软件构件通常包括问题(及子问题)描述、有关的论域知识、相应的需求规格说明。问题描述及论域知识均使用论域语言，必要时可辅以结构化的自然语言。

类似地，设计级软件构件包括设计规格说明(片段)、设计决策描述、相应的设计文档(如总体设计方案、抽象算法)。这时，若能将需求级的软件构件和设计级的软件构件相结合，使需求级软件构件和设计级软件构件的复用同时发生，就能够简化对这些软件构件的理解和组装过程。 需求级和设计级软件构件的开发必须遵循以下两条原则：

§ 将构件应用的上下文与构件严格分离；

§ 利用抽象化、参数化等手段提取公共特征，增强构件对未来不同应用项目的适应能力。

需求级、设计级与代码级软件构件相结合，有助于代码级软件构件的提取和理解。借助较高抽象级别的软件构件可望提高复用的效能，因为这样不仅能够复用代码，而且能够复用需求模型和设计方案。

(2) 代码级的软件构件

代码级复用是迄今为止研究最深入、应用最广泛的复用技术。代码级软件构件不仅应包含通常的程序代码段，还应辅以相应的注释和上下文说明。这些辅助可以用结构化语言或其它标识机制描述，但最好以相应的需求级或设计级软件构件的形式出现。传统上，代码级软件构件的主体是子程序(过程或函数)。现在，在基于对象和面向对象程序设计语言及相应的开发环境下，软件构件可以用 Ada 程序包或 C++类来实现。

开发代码级软件构件与通常的程序设计活动不同，其主要区别在于：软件人员必须运用各种抽象化手段，以发掘公共特征，分离构件的应用上下文、功能和行为，从而提高可复用性。典型的抽象方法有如下 3 种。

§ 功能抽象：软件构件的功能由接口说明中的输入－关系确定，实现细节对构件的使用者隐藏起来；

§ 数据抽象：在功能抽象的基础上进一步隐藏除接口参数外的所有数据。构件的功能或行为由输入参数和构件自身记忆的内部状态决定。构件的内部状态可由构件的内部操作来更新。面向对象程序设计语言中的“类”是一种典型的基于数据抽象的软件构件。

§ 过程抽象：在数据抽象的基础上进一步实现在同一软件构件上并发执行的多个线程的无关性。软件构件提供端口，以便为多个用户同时访问软件构件中的资源进行同步控制。访问请求首先进入等待队列，软件构件就绪后，从队列中取出请求逐个执行。使用构件的多个线程之间通过全局共享数据或消息传递进行信息交换。软件构件的设计必须保证构件接口元素的功能和行为在语义上的正确性，不管各线程操作序列之间的相对时序如何变化。

(3) 程序设计的主要原则

可复用性、可理解性、正确性和易组装性是衡量软件构件质量的重要标准。这些标准都与构件开发过程中的程序设计风格密切相关。对于代码级软件构件的开发者，必须注意遵循以下的程序设计原则。

① 可复用性与易组装性

§ 抽象化、参数化、模板化；

§ 显式建立构件的外部接口，保证接口在语法上和语义上的清晰性；

§ 除参数外，接口中的成分不应与运行环境相关；

§ 使用构件的信息应与构件的实现细节相分离；

§ 构件接口的使用方式、功能和行为模式应遵循论域分析所确立的公共标准。

② 正确性

§ 使用显式的、标准的、完备的出错处理机制；

§ 使用强类型的程序设计；

§ 在构件上附加重要的测试数据和测试报告。

③ 可理解性

§ 提供完全的、精确的文档和程序注解；

§ 构件内部的结构、数据和控制流程必须标准化；

§ 使用统一的命名规则和语法约定。

(4) 软件构件的质量保证 由于软件构件将在多种硬件和软件环境下运行，因此，除采用软件工程的一般的质量保证措施之外，还要针对可复用构件的特征进行特别的质量保证。

① 构件的开发者应当利用现有的硬件、软件，在尽可能多的环境中进行各种标准测试。但是。由于条件的限制及未来运行环境的不可穷举性，这种方法对于构件的质量保证并不充分，必须与其它方法配合使用。

② 在构件的设计过程中必须预先考虑构件对于各类可能的硬件环境的可移植性和对于不同软件环境的适应能力。根据设计制定相应的编程原则，并在编程的过程中贯彻始终。

③ 应当尽可能将构件的与环境有关的因素抽象成为构件的参数，或者可将这些因素作为使用构件的约束条件在接口说明中详细阐述，甚至可以将这些约束条件标识成精确的逻辑断言插装到构件的源程序中。

④ 对可能引发移植性错误和适应性错误的出错源进行分类，针对每一类出错源制定相应的防范和测试措施。移植性错误包括数值计算错误、多任务调度错误、内存使用错误等；适应性错误包括并发性错误、重名错误、表达式求值顺序错误等。针对每一类出错源制定相应的防范和测试措施与硬件、软件环境密切相关。

⑤ 制订并执行统一的、良好的程序设计风格。这对于确保构件质量，提高可复用性、可理解性和易组装性有重要的意义。

⑥ 成立构件质量保证小组。其主要任务是：制订构件合格的标准和构件质量保证计划，对构件开发过程进行监控，保证上述措施的落实。

5. 可复用构件库的组织

需要对收集和开发的软件构件进行分类，然后把它们放入到可复用构件库的适当为止。软件构件的分类方法和相应的构件库结构对构件的检索和理解有极为深刻的影响，就是说，可复用构件库的组织应当便于构件的存储和检索。

复用技术对可复用构件库组织方法的要求是：

§ 支持构件库的各种维护操作。增、删、更新构件库应当尽量不影响构件库的结构；

§ 不仅能够支持精确匹配，还应能支持相似构件的查找；

§ 不仅能够进行简单的语法匹配，而且能够查找在功能和行为上等价或相似的构件；

§ 对应用论域(族)有较强的描述能力和较好的描述精确度；

§ 便于库管理员和用户使用；

§ 具备可自动化的潜力。

(1) 枚举分类

此方法通过定义一个层次结构类描述构件。在这个层次中定义软件构件的类以及不同层次的子类。真实的构件都被罗列在枚举层次中任一路径的最低层。例如，对窗口操作的枚举层次可能是：

window operations

display

open

menu-based

openWindow

system-based

sysWindow

close

via pointer

……

resize

via command

setWindowSize, stdResize, shrinkWindow

via drag

pullWindow, StretchWindow

up/down shuffle

……

move

……

枚举分类模式的层次结构使得它易于理解和使用。但在建立层次之前，必须进行论域工程，这样层次中各个项可以有充足的知识以供使用。

(2) 关键词分类

此方法的基本思想是：根据论域分析的结果，将应用论域(族)的概念按照从抽象到具体的顺序，逐步分解树形结构或有向无回路图结构。每个概念用一个描述性的关键词表示。不可再分解的原子层的包含隶属于它的某些软件构件。图 8.3 给出了可复用构件库的关键词分类结构，它支持图形用户界面设计。

当加入新的构件时，库管理员必须对构件的功能或行为进行分析，在浏览上述关键词分类结构的同时，将构件置于最合适的原子层关键词之下。如果无法找到构件应从属的关键词，可以扩充现有的关键词分类结构，引进新的关键词。但库管理员必须保证新关键词有相应的论域分析的结果作为支持。

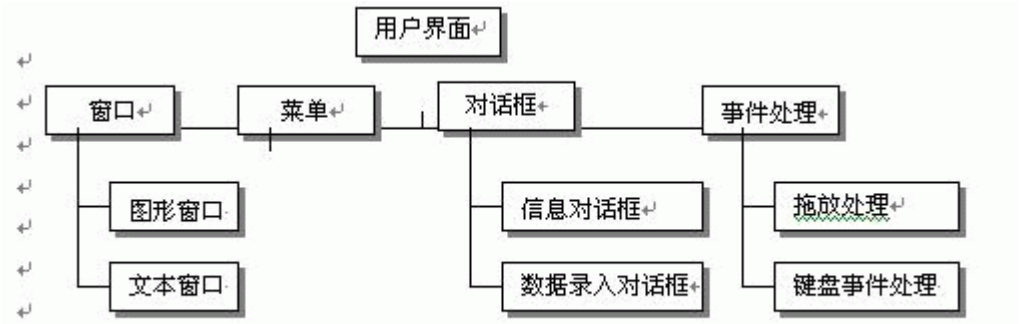


图 8.3 关键词分类结构示例

(3) 多面分类

此方法由 3 部分构成：多面分类机制、同义词库和概念距离图。

① 多面分类机制：分析论域范围并定义若干用于描述一个构件特征的“面”，每个“面”包含若干“概念”，它们表述构件在“面”上的基本特征。这些特征根据它们的重要性排队。“面”可以描述构件执行的功能、被操作的数据、构件应用的上下文以及任何其它特征。描述某一构件的“面”的集合称为面描述子。通常，限制面的描述不超过 7 到 8 个面。

作为一个简单的在构件分类中使用“面”的例子,面描述子的模式可以是：｛ 功能，对象类型，系统类型 ｝

面描述子中每一个“面”可含有 1 个或多个特征值,这些值一般是描述性的关键词。例如，若功能(function)是某一构件的一个面，赋与此面的典型值可能是：

function = (copy, from) or (copy, replace, all)

使用多重面特征值可使原始的函数 copy 充分地细化。

② 同义词库：意义相同或相近的若干词汇组成同义词库。所有词汇按照隶属于“面”的“概念”分组，在任一时刻点，每个“概念”可用组内的某一同义词汇作为标识载体。

③ 概念距离图：用于度量每个“面”中“概念”的相似性程度。属于每个“面”的一般化概念与其中的两个或多个“概念”以加权边相连接，两个“概念”的相似性由它们之间的最短加权路径上的加权距离确定，附加于边上的权值体现了“概念”之间的差异程度。

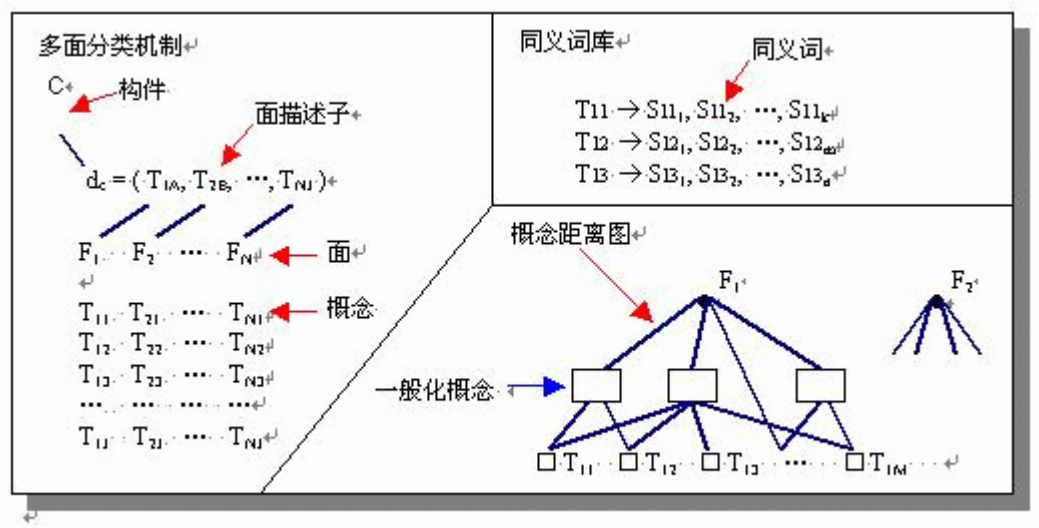


图 8.4 多面分类法的组成

采用多面分类法进行可复用构件库的组织，必须在存储软件构件的同时，表示并存储多面分类机制、同义词库和概念距离图。例如，可以采用关系数据库中的表格来描述它们。多面分类法的所有语法构件(“面”、“概念”、同义词、一般化概念、差异性权值)均取材于论域分析的结果。当需要在可复用构件库中加入新的构件时，库管理源必须对构件的功能、行为进行深入分析，利用现有的多面分类结构确定构件的描述子，对每个“面”选取合适的“概念”作为特征描述。必要时可以考虑增加新的“概念”，此时必须根据新“概念”完善同义词库和概念距离图。

(4) 超文本组织法

超文本方法与基于数据库系统的可复用构件库组织方法不同，它基于全文检索技术，其基本思想是：所有软件构件都必须辅以详细的功能或行为说明文档，说明中出现的概念和软件构件以网状链接方式互相连接。检索者在阅读文档的过程中可按照人类的联想思维方式任意跳转到包含相关概念或软件构件的文档中去。全文检索系统将用户给出的关键词与说明文档中的文字进行匹配，实现软件构件的浏览式检索。

超文本是一种非线性的网状信息组织方法，它以结点为基本单位，链作为结点之间的联想式关联，如图 8.5 所示。一般地，结点是一个信息块。对于可复用构件库而言，结点可以是论域的概念、功能或行为名称、构件名称等，在图形用户界面上，结点可以是字符串，也可以是图像、声音、动画等。超文本组织方式为构造可复用构件提供了友好、直观的多媒体方式。由于网状结构比较自由、松散，因此，超文本方法更容易修改可复用构件库的结构。

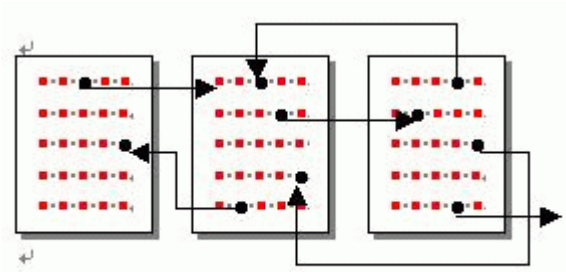


图 8.5 超文本结构示意图

Windows 环境下的联机帮助系统就是一种典型的超文本系统。为了构造可复用构件库的文档，首先要根据论域分析的结果，在说明文档中标识超文本结点，并在相关文档中建立链接关系。然后用类似于联机帮助系统编译器的工具对构件的说明文档进行编译，最后用相应的工具运行编译后的目标程序代码即可。

(5) 可复用构件的 3C 模型

可以有很多方法来描述可复用的构件，其中最理想的是由 Tracz 提出的 3C 模型：概念(Concept)、内容(Content)和上下文(Context)。

§ 概念：描述软件构件做什么。为此，需要完全地描述构件的接口和表示语义(表示在前置条件和后置条件的上下文中)。概念将传达构件的意图。

§ 内容：描述软件构件的概念如何实现。一般来讲，内容是对外来用户隐蔽的信息，是只有试图修改或测试该构件的人才需要了解的信息。

§ 上下文：将可复用的构件安置到它的应用论域中。就是说，上下文将通过定义概念的、操作的和实现的特征，使软件人员能够找到合适的构件以满足应用的需求。

为了在实际背景下使用，必须把概念、内容和上下文转换成具体的规格说明模式。关于可复用构件的分类模式，已有不少文章讨论过。所有的方法大致归为 3 个主要的方面：图书馆和信息科学方法、人工智能方法和超文本系统。目前，绝大多数研究都建议使用图书馆科学方法为软件构件进行分类。图 8.6 给出了一个使用图书馆科学索引法的分类法。“受控的索引词汇表”限制了所有可用于分类对象(或构件)的术语或语法。“不受控的索引词汇表”则对描述的自然状态不加限制。软件构件的分类模式主要有如下 3 类：

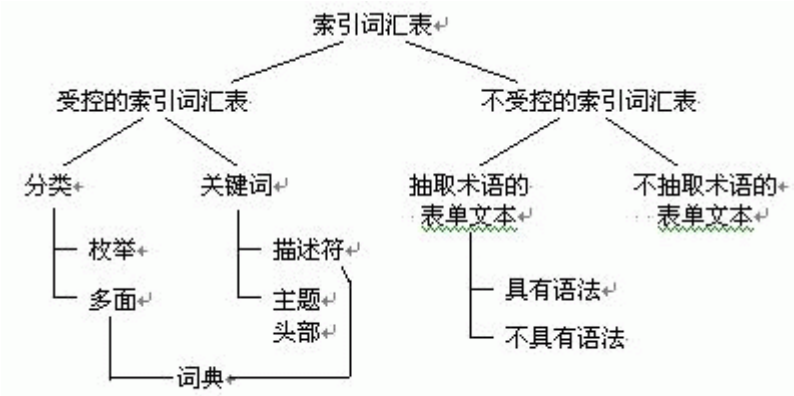


图 8.6 源于图书馆索引方法的分类法

6. 分层式体系结构

所谓分层式体系结构，是按层次组织软件的一种软件体系结构，其中每一层软件建立在低一层的软件层上。位于同一层的软件系统或子系统具有同等的通用性，在下一层的软件比在上一层的软件通用性更强。一个层次可视为同等通用档次的一组(子)系统。

因此，在分层的体系结构中，最高层是应用层，可包容许多应用系统。次高层是构件层，可包括多个可复用构件库系统，可用于建立应用系统。应用系统建立在构件层之上，而此构件层中的许多构件库系统又是建立在更低层次的构件库系统之上。

软件的组织方式通常是指软件的静态分层结构，就像在编译连接时软件各个模块之间的分层依赖关系那样，是一种静态的关系，而不是指软件在运行时的组织和动态结构。一个系统的动态特征时由使用事例、协作、过程和结点模型来定义的。这些动态模型需要与软件的静态分层组织联合起来使用。即使按照上述的原则，人们仍然定义出多种形式的分层式体系结构。层次的数目、层次的名称、层次的内容可随情况而定。图 8.7 给出一种典型的 4 层次体系结构。

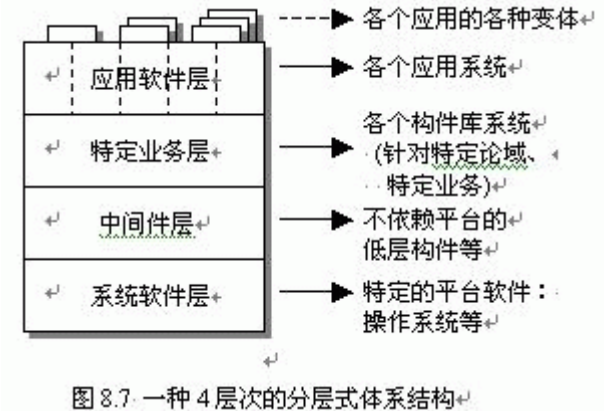


图 8.7 一种 4 层次的分层式体系结构

最顶层(即最高层)是应用系统层，此层包含多个应用系统，每个应用系统向系统的用户提供一组使用事例。有的应用系统还可具有不同的版本或若干变体。应用

系统可以通过其接口与其它系统操作，还可以通过低层软件提供的服务或对象(如操作系统、特定业务服务)间接地与其它系统交互操作。

次顶层(即次高层)是“特定业务”层。此层应当包括专门针对不同业务类型的一系列构件库系统。这样的构件库系统向用户提供可复用的使用事例和对象构件，用于开发应用系统，支持复用业务。特定业务层的软件建立在中间件之上。 中间件层位于次高层下面，它为次顶层的各个构件库系统提供实用软件类，以及不依赖于平台的服务。例如，在异种机型环境下的分布式对象计算等等。此层经常包括:图形用户界面构筑者使用的构件库系统、与数据库管理系统(DBMS)的接口、不依赖平台的操作系统服务、对象请求代理(ORBs)、对象链接和嵌入 OLE 构件，如电子表格和框图编辑器。这些软件主要提供给应用软件和构件开发人员使用，使得他们能够专注于业务构件和应用系统的构筑。

最低层是系统软件层，此层包括计算和网络等基础设施软件，如操作系统、专用的硬件接口软件等。

目前出现了一些专用操作系统，其本身就提供了不依赖于平台的服务，因此，在依稀结构的第三层和第四层之间，有时界限会变得模糊不清。例如 Java 就是这种情况。它是一种语言，故它应该位于系统软件层。但还可以把 Java 看作是组织分布对象的一个重要部分，通过 Java 可将对象移到不同的机器上，从而改变客户机－服务器系统的应用划分。从另一个角度来看，Java 的一个重要部分又属于中间件层，至少位于中间件层的许多软件都是用 Java 语言编写的。

为了确保分层式系统可管理，规定在一个系统内，不能从低层复用高层的构件。一个分层式系统有两维：水平方向是在同层次内的相互引用的多个系统，垂直方向表达了跨层次的静态的依赖关系。

7. 软件构件的复用

(1) 检索与提取构件

可复用构件库的检索方法与库的组织方式密切相关。

① 基于关键词的检索

这种检索方法的基本思想是：系统(CASE 工具)在图形用户界面上将可复用构件库的关键词树直观地展示给用户，用户通过在树上的逐级浏览寻找需要的关键词并提取相应的构件。当然，用户也可以直接给出关键词(其中可含有通配符)，由系统自动地给出合适的候选构件清单。

这种方法的优点是简单，易于实现。

② 多面检索

这种检索方法基于多面分类法。步骤如下：

§ 构造查询：用户提供待查构件在每个“面”上的特征，生成构件描述子。此时，用户可以从可复用构件库已有的“概念”中挑选，也可以将某些特征值置为空。系统在库中检索时将忽略特征值为空的“面”。在构造查询的过程中，可以利用同义词库和概念距离图来帮助用户正确选择特征值。

§ 检索构件：实现多面分类法的 CASE 工具利用同义词库和概念距离图，在可复用构件库中寻找相同的或相近的构件描述子及相应的构件。

§ 对构件进行排序：按照相似程度对被检索出来的构件进行排序，还可以按照与复用有关的度量信息(如构件的复杂性、可复用性、成功复用的次数等)进行排序。

这种方法的优点是易于实现相似构件的查找。

③ 超文本检索

这种检索方法的步骤是：用户首先给出一个或若干个关键词，系统在构件的说明文档中做精确的或模糊的语法匹配。匹配成功后，向用户提供相应的构件说明。这些构件说明是含有许多超文本结点的正文。用户在阅读这些正文时可实现多个构件说明文档之间的自由跳转，最终选择合适的构件。为了避免用户在跳转过程中迷失方向，系统可以通过图形用户界面显示浏览历史图，允许将特定画面定义为有名“书签”并可随时跳转到“书签”。此外，还可以帮助用户逆着跳转路径逐步返回。 这种方法的优点是用户界面友好。

④ 其它检索方法

上述检索方法都是基于语法匹配，要求用户对可复用构件库中的出现的众多词汇有较全面的把握，较精确的理解。理论上，理想的检索方法是语义匹配：可复用构件库的用户以形式化的手段描述所需要的构件的功能或行为的语义，系统通过定理证明或基于知识的推理过程寻找语义上等价的或相近的构件。

(2) 理解与评价构件

准确地理解构件，对于正确地使用和修改构件，都是至关重要的。考虑到设计信息对于理解构件的必要性和构件库的用户逆向发掘设计信息的困难性，要求构件的开发过程必须遵循公共的软件工程规范，并在构件库的说明文档中全面、准确地说明下列内容：

① 构件的功能和行为；

② 相关的论域知识；

③ 可适应性约束条件和例外情况；

④ 可以预见的修改部分和修改方法。

但是，如果软件人员希望复用那些原先并非为复用而设计的构件，上述要求不能满足。此时软件人员必须借助 CASE 工具对待选构件进行分析。这种 CASE 工具对构件(包括相应的文档)进行扫描，将各类信息存入某种浏览数据库，然后回答构件用户的各种查询，进而帮助理解。例如，对于以 C 或 C++ 源程序形式出现的构件，用户可借助于 CASE 工具查询标识符(包括变量、常量、宏、函数等)的定义和引用。如果源程序中以结构化注解的形式给出函数的前置条件、后置断言、约束与例外情形、算法描述，用户也可以实现对这些信息的直观浏览。

逆向工程是理解构件的另一种重要手段。通过对构件进行分析，结合论域知识，半自动地生成构件的设计信息，然后借助设计信息完成对构件的理解和修改。

对软件构件的可复用性进行评价，主要通过收集和分析构件的用户在实际复用构件的过程中所得到的各种反馈信息，按照某种论域模型来完成。这些反馈信息包括：复用成功的次数、对构件的修改工作量、构件的健壮性度量(如出错数量)、性能度量(如执行效率和资源消耗量)等。

(3) 修改构件

理想的情况是对库中的构件不做修改就可以直接用于新的软件项目。但是，在多数情况下，需要对构件做或多或少的修改以适应新的需求。为了减少修改的工作量，要求构件的开发人员尽量使构件的功能、行为、接口抽象化、通用化、参数化。这样，构件的用户可以通过对实参的选择来调整构件的功能或行为。如果这种调整仍不能使构件适应新的软件项目，用户就必须借助设计信息和说明文档来理解、修改构件。因此，与构件有关的说明文档和抽象层次更高的设计信息对于构件的修改至关重要。例如，如果需要将用 C 语言编写的构件(源程序代码)改写为其它语言的形式，构件的算法描述就十分必要。

有人在软件复用领域中尝试使构件的修改自动化。例如，可以利用不同数据结构(如链表和数组)之间的相似性自动修改构件外部接口中的参数类型。但是，这种自动修改方式还远远不能取代人工修改方式。

(4) 构件的合成

构件合成是指将可复用构件库中的构件(经适当修改后)相互连接，或将它们与当前软件项目中的软件元素相连接以构成最终的目标系统。构件合成技术大致可分为基于功能的、基于数据的和面向对象的合成技术。下面简单介绍前两种。

① 基于功能的合成技术 基于功能的合成技术采用子程序调用和参数传递的方式将构件结合起来。它要求在库中的构件必须以标准子程序(标准过程或函数)的形式出现，并且接口说明必须准确、清楚。当使用这种合成技术进行软件开发时，必须开发人员必须对目标软件系统进行自顶向下的功能分解，将系统分解为高内聚、低耦合的功能模块，然后根据各模块的功能需求提取构件，对它们进行适应性修改后，再纳入到上述功能分解的层次框架中。

② 基于数据的合成技术

基于数据的合成技术首先根据当前应用问题的核心数据结构设计出一个框架，然后根据框架中各结点的需求提取构件并进行适应性修改，再把它们逐个分配给框架中的适当位置。此后，构件的合成方式仍然是传统的子程序调用与参数传递。这种合成技术也要求库中的构件以子程序的形式出现，但它所依赖的软件设计方法不再是功能分解，而是面向数据结构的设计方法，如 Jackson 系统开发方法。

8. 面向对象的软件复用技术

由于封装和继承的特性，面向对象方法比其它软件开发方法更适合支持软件复用。封装意味着可以将表示构件的类看作黑盒子。使用它们时，只需了解其外部接口，即了解它能够响应哪些消息，相应的对象行为是什么。继承是指在定义新的子类时可利用库中已有的父类的属性和操作。当然，子类也可以修改父类的属性与操作，或者引进新的属性与操作。理论上构件的用户不需要了解构件的实现细节。

(1) 类库的构造

通常将面向对象的可复用构件库称为可复用类库(简称类库)，因为这时所有的构件都是以类的形式出现。可复用基类的建立取决于论域分析阶段对当前应用(族)中具有一般适用性的对象和类的标识。类库的组织方式采用类的继承层次结构。这种结构与现实问题空间的实体继承关系有某种自然、直接的对应。同时，类库的文档以超文本方式组织，每个类的说明文档中都可以包含指向其它说明文档的关键词结点的链接指针。

与图形用户界面有关的一种典型的类库结构如图 8.8 所示。

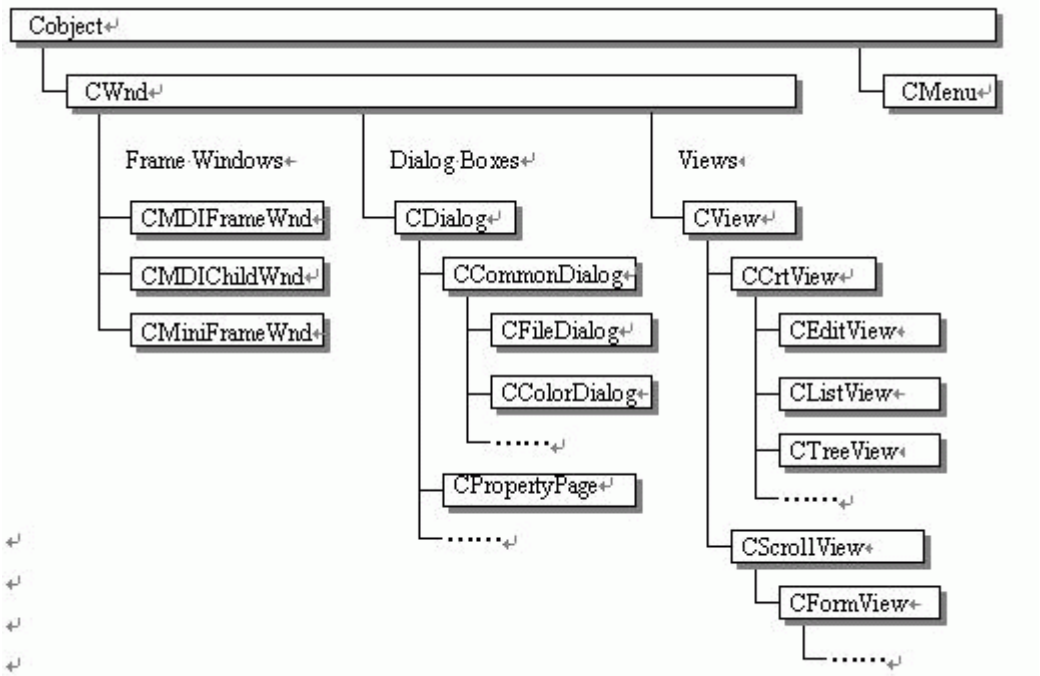


图 8.8 类库结构示例

(2) 类库的检索

一般而言，类库的组织方式直接决定检索方式。常用的类库检索方法是对类库中类的继承层次结构进行树形浏览，以及基于类库文档的超文本检索。借助于树形浏览工具，类库的用户可以从树的根部(继承层次的根类)出发，根据对可复用基类的需求，逐层确定它所属的语法和语义的范畴，然后确定最合适的基类。借助于类库的超文本文档，用户一方面可以在类库的继承层次结构中查阅各基类的属性、操作和其它特征，另一方面可按照基类之间的语义关联实现自由跳转。

需要强调的是：对类库的检索并不要求待实现的类与库中的基类完全相同或极为相似，只是希望待实现的类与基类之间存在某种自然的继承关系，或者基类能够提供属性或操作给待实现的子类选用。这与其它可复用构件库的检索截然不同。

(3) 类的合成

如果从类库中检索出来的基类能够完全满足新软件项目的需求，则可以直接复用。否则必须以类库中的基类为父类，采用构造法或子类法派生出子类。注意，面向对象的复用技术通常不允许用户修改库中的基类，要想对类库进行扩充或修改，应当调整类库的继承结构以把新的子类加入到适当的位置。

① 构造法：为了在子类中使用库中的基类的属性和操作，可以考虑在子类中引进基类的实例作为作为子类的实例变量，然后在子类中通过实例变量来复用基类的属性或操作。构造法只用到面向对象的封装特征。

② 子类法：与构造法完全不同，子类法把新子类直接说明为库中基类的子类。通过继承和修改基类的属性和操作来完成新子类的定义。子类法利用了面向对象的封装和继承的特性。

9. 支持软件复用的 CASE 工具

能否在软件开发过程中成功地使用复用技术，直接取决于 CASE 工具对软件复用的支持程度。下面主要介绍复用技术的各个子任务对于 CASE 工具的大致需求。当然，这些 CASE 工具不是软件复用的必要条件，但具备这些工具能提高复用的效能。

(1) 论域分析

表 8.3 论域分析的关键子任务及其支持工具

关键子任务	支持工具
知识获取	专家系统建造工具
对象及其操作的标识	实体－关系图工具
抽象与关联	面向对象的开发工具
对象分类	基于语义的自动分类工具
论域语言处理	语法分析工具

(2) 构件的开发

表 8.4 软件构件开发的关键子任务及其支持工具

关键子任务	支持工具
提高构件独立性	依赖性分析工具(如交叉引用生成器)，软件结构分析器(如调用关系生成器、继承关系生成器)
参数化	程序设计语言中的宏替换，编译器中的预处理器，类属机制(如 Ada 中的 Generic)
抽象与特殊化	面向对象语言的继承机制
构件的扩充(增加功能，扩大适用范围)	面向对象的语言机制，支持同一软件元素以多个版本不断演化的配置管理工具
测试与认证	测试覆盖面分析工具
形式验证	定理证明工具
构件质量评估	软件质量的度量与分析工具
构件的分类	语义近似性分析工具，基于规则的专家系统

(3) 构件库的组织与检索

表 8.5 软件构件库组织与检索的关键子任务及其支持工具

关键子任务	支持工具
可复用构件库的组织与存储	关系数据库或面向对象数据库管理系统，类库
浏览	超文本浏览工具
检索	自然语言的语法分析与语义理解工具
结构化查询	关系数据库管理系统
基于语义的检索	定理证明工具(用于语义匹配)

(4) 构件的合成

表 8.6 软件构件库合成的关键子任务及其支持工具

关键子任务	支持工具
构件的修改	源程序代码的比较工具
构件的实例化	宏扩展工具，模板程序的实例数据生成工具
修改后构件的正确性验证	用于支持在软件维护阶段修改软件元素的 CASE 工具
将构件合成到目标软件系统	编译器，连接器以及集成化 CASE 环境中的其它合成机制

三、例题分析

【例 1】实施软件复用的目的是要使软件开发工作进行得(A)。软件复用的实际效益除了(B)之外，在企业的经营管理方面也可望达到理想的效益。

新的应用软件开发技术和工具是以(C)作为关键，复用大粒度的(D)，为的是快速开发应用软件。这些新技术包括微软的(E)、(F)、(G)，SUN 公司的 Java，OMG 公司的 CORBA、IDL 等。

供选择的答案：

A. ① 更简捷 ② 更方便 ③ 更快、更好、更省 ④ 更丰富

B. ① 复用率 ② 功能扩充 ③ 效率 ④ 空间利用率

C, D. ① 软件 ② 固件 ③ 构件 ④ 属性

⑤ 对象 ⑥ 事物 ⑦ 数据 ⑧ 代码

E~G. ① office ② Visual Basic ③ Active X ④ Photoshop

⑤ OLE ⑥ Fortran ⑦ COBOL ⑧ Delphi

答案：A. ③, B. ①, C. ③, D. ⑤, E. ②, F. ③, G. ⑤。其中，E、F、G的答案顺序可互换。

分析：实施软件复用的目的是要使软件开发工作进行得是更快、更好、更省。“更快”是指在市场竞争环境中，软件开发工作能满足市场上时间方面的要求(即在提供软件产品的时间方面能赛过竞争对手)；“更好”是指开发出来的软件在未来的运行中失效可能性小；“更省”是指在开发和维护期间所花费的开销少。

日美一些大公司的资料表明，软件复用率最高可望达到90 %，而且软件复用使得企业在及时满足市场、软件质量、软件开发和维护费用等方面都得到显著的改进。

除了复用率之外，在企业的经营管理方面也可望达到理想的效益。例如，上市时间可缩短2 ~ 5倍；软件产品的缺陷密度可减少5 ~ 10倍；软件产品的维护费用可减少5 ~ 10倍；软件开发总费用可减少15% ~ 75%，其中，75%是针对长期项目，包括开发可复用构件及支持复用的负担。

新的应用软件开发技术和工具是以“构件”作为关键，复用大粒度的“对象”，为的是快速开发应用软件。这些新技术包括微软的Visual Basic、Active X、OLE(对象链接与嵌入)，SUN公司的Java，OMG公司的CORBA(公用对象请求代理程序体系结构)、IDL(接口定义语言)等。非面向对象语言(如COBOL和Fortran)在复用实践中已经器的相当的成功。这些非面向对象程序设计语言构件技术的成功实践说明了：实现软件复用并不限于面向对象语言构件或类库。

【例2】以往的软件工程技术不能满足复用的需要，体现在工程、(A)、(B)、经营业务等4个方面。“工程”指软件开发工程,表现在缺乏(C)手段,缺乏(D)构件，缺乏对潜在可复用的(E)，缺乏实施复用的工具。

供选择的答案：

A, B. ① 需求 ② 过程 ③ 环境 ④ 组织管理

C ~ E. ① 复用 ② 可靠性 ③ 灵活性 ④ 界定

⑤ 工具 ⑥ 互连性

答案：A. ②, B. ④, C. ④, D. ①, E. ③。其中，A、B 的答案的顺序可互换。

分析：以往的软件工程技术不能满足复用的需要，体现在工程、过程、组织管理、经营业务等 4 个方面。这里所说的“工程”是指软件开发工程，其技术和方法面对复用的需要已显得低效，主要表现在：

§ 缺乏界定手段：为了软件复用，需要循软件开发流程的各个阶段，通过分析它们的描述模型，明确界定出潜在可复用的部分，被界定出的部分，可能是可被复用的部分，也可能是可被可复用构件代用的部分。而以往的软件工程缺乏这种界定手段。

§ 缺乏可复用的构件：者反映在许多方面。例如，不能有效地挑选出可复用构件并对它们进行强化；缺乏对构件打包、文档化、分类、界定的技术；缺乏有效方法进行(构件)库的设计和实现；缺乏良好的构件库存取方法。

§ 缺乏对潜在可复用构件的灵活性：如果一个构件很死板，那么它被复用的机会就很少。而过去的软件工程方法在设计灵活的、分层的软件体系结构方面一直没有不成熟的办法。过去的方法是对构件进行调节使之满足新的需求，或者使对新的体系结构进行限制。

§ 缺乏实施复用的工具：为了实施复用，需要一系列新的工具，并把它们集成到面向复用的软件工程环境中去。而过去的工程缺乏这方面的工具。

【例 3】论域工程过程要在选定的应用论域中界定出(A)和(B)，要为多个应用和构件定义一个(C)，并开发一系列可适度扩展的(D)。创建可复用构件既困难又昂贵。所以，应当帮助软件人员进行界定工作，以及按重要性对各项(E)进行优先性排队。在一定程度上，这项工作要依赖于(F)。

供选择的答案：

A, B, E, F：① 共性 ② 特性 ③ 可变性 ④ 属性

⑤ 灵活性 ⑥ 可靠性 ⑦ 健壮性 ⑧ 适用性

C, D： ① 固件 ② 构件 ③ 体系结构 ④ 组件

⑤ 总体框图 ⑥ 数据结构

答案：A. ①, B. ③, C. ③, D. ②, E. ②, F. ⑥。其中，A、B 的答案顺序可互换。

分析：论域工程过程要在选定的应用论域中界定出共性和可变性，要为多个应用和构件定义一个体系结构，并开发一系列可适度扩展的构件。创建可复用构件既困难又昂贵。所以，应当帮助软件人员进行界定工作，以及按重要性对各项“特性”进行优先性排队。在一定程度上，这项工作要依赖于预测的可靠性(预测需要哪些应用层的可靠性和构件层的可靠性)，这也关系到我们要冒多大的风险。

【例 4】软件体系设计的一个中心问题是能否(A)，以及采用何种软件体系结构风格。有原则地使用体系结构风格可带来一些实际的好处：

(1) 它促进了对设计的(B)；

(2) 它可以带来显著的(C)(体系结构风格的不变部分使它们可以共享同一个实现代码)；

(3) 只要系统是使用常用的、规范的方法组织起来的，就可以让其它设计者很容易地理解软件的体系结构；

(4) 对标准或规范风格的使用也支持了(D)，例如像 CORBA 这样的面向对象的架构和基于事件机制的工具的集成；

(5) 在限制了设计空间的情况下体系结构风格通常允许进行特殊的与风格有关的分析；

(6) 通常可以对特定的风格提供(E)手段。

供选择的答案：

A ~ C. ① 使用特定的体系模式 ② 使用重复的体系模式

③ 复用 ④ 专用 ⑤ 代码复用 ⑥ 结构复用

D ~ E. ① 可视化 ② 构件 ③ 互操作性 ④ 可移植性

⑤ 可调度性 ⑥ 框架

答案：A. ②, B. ③, C. ⑤, D. ③, E. ①。

分析：软件体系设计的一个中心问题是能否使用重复的体系模式，以及采用何种软件体系结构风格。例如可以采用通用的基于层次或数据流的系统体系结构，或者采用特殊的系统组织(如经典的编译器分解方式)，OSI 七层协议，MVC 用户界面语义图等。有原则地使用体系结构风格可带来一系列实际的好处。

(1) 它促进了对设计的复用。一些经过实践证实的解决方案可以拿来可靠地解决新问题。

(2) 它可以带来显著的代码复用。体系结构风格的不变部分使它们可以共享同一个实现代码。

(3) 只要系统是使用常用的、规范的方法组织起来的，就可以让其它设计者很容易地理解软件的体系结构。例如，如果某人把系统描述为“客户机／服务器”模式，则不必给出细节，人们立刻就会明白它们是如何一部分一部分地组织起来，并在脑海中清晰地得到这个图像。

(4) 对标准或规范风格的使用也支持了互操作性，例如像 CORBA 这样的面向对象的架构和基于事件机制的工具的集成；

(5) 在限制了设计空间的情况下，体系结构风格通常允许进行特殊的与风格有关的分析。例如，分析管道过滤器系统的可调度性(吞吐量、延时、死锁的解决)是可行的，但对于任意的、或用其它方法构造的架构来说，这种分析可能毫无意义。

(6) 通常可以对特定的风格提供可视化的手段。例如，可以对与客户的专业领域有关的设计作出图解或文字说明。

对于应用软件体系结构风格来说，由于视点的不同，软件人员有很大的选择空间。要为系统选择或设计某一个体系结构风格，必须根据特定项目的具体特点，进行分析比较后再确定。体系结构风格的使用几乎完全是特化的。

【例 5】分层系统采用层次化的组织方法，每一层向其(A)提供服务，并利用(B)的服务。在一些分层系统中，(C)全部被隐藏起来，只有(D)和一部分精心选择的功能可以被系统外部看到。在这种系统中，(E)是实现在层次结构中的一些虚拟机，(F)是层次与层次之间交互的协议，(G)包括对层次之间交互的限制。分层系统中有许多可取的属性。首先，它支持(H)的系统设计，这使得设计者可以把一个复杂的系统按递增的步骤分解开来；其次，它支持(I)，像管道结构的系统一样，因为每一层至多和相邻的上下层交互，因此，功能的改变最多只影响相邻的上下层。另外，它能支持(J)，和抽象数据类型一样，只要提供的服务接口定义不变，同一层的不同实现可以交换使用。 供选择的答案：

A ~ D. ① 下层 ② 内部层次 ③ 外部层次 ④ 上层

⑤ 中间件 ⑥ 接口

E ~ G. ① 接口定义 ② 软件部件 ③ 连接 ④ 拓扑约束

⑤ 规范定义 ⑥ 推理机制 ⑦ 系统结构

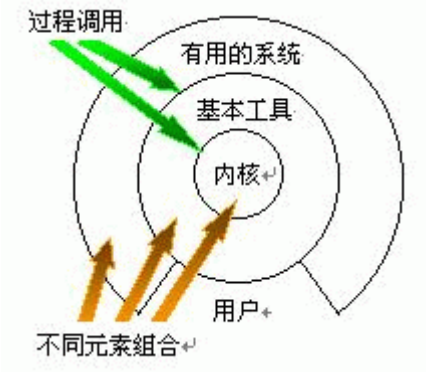
H ~ J. ① 基于抽象程度递增 ② 基于具体细节递增

③ 复用 ④ 可视化 ⑤ 互操作 ⑥ 功能增强

⑦ 效率提高 ⑧ 结构化

答案：A. ④，B. ①，C. ②，D. ③，E. ②，F. ③，G. ④，H. ①，I. ⑥，J. ③。

分析：



对于分层系统：

(1) 基本结构：分层系统采用层次化的组织方法，每一层向其上层提供服务，并利用其下层的服务。在一些分层系统中，内部层次全部被隐藏起来，只有外部层次及一部分精心选择的功能可以被系统外部所见。在这种系统中，软件部件是实现在层次结构中的一些虚拟机，连接是层次与层次之间交互的协议，拓扑约束包括对层次之间交互的限制。如图所示。

(2) 应用：这种系统最广泛的应用是分层通信协议。在这一应用论域中，每一层提供一级抽象的功能，作为上层通信的基础。较低的层次定义低层的交互，最低层通常只定义硬件物理连接。其它应用论域有数据库系统、操作系统等。

(3) 分层系统中有许多可取的属性。首先，它支持基于抽象程度递增的系统设计，这使得设计者可以把一个复杂的系统按递增的步骤分解开来；其次，它支持功能增强，像管道结构的系统一样，因为每一层至多和相邻的上下层交互，因此，功能的改变最多只影响相邻的上下层。另外，它能支持复用，和抽象数据类型一样，只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，并允许有各种不同的实现方法(典型的例子是 ISO OSI 参考模型和某些 X Windows 的系统协议)。

【例 6】软件复用可分为以下三个层次：

(A)，例如软件工程知识的复用。

(B)，例如面向对象方法或国家制定的软件开发规范的复用。

软件成分的复用。软件成分的复用又可进一步划分为(C)，如剪贴；(D)，如复用设计模型；(E)，更高级别复用。

为了研究软件复用程度与软件生产率之间的关系，我们引进：生产率 P = (F) 和复用率 R = (G)的公式，以及两者之间的关系 P = (H)。其中，用 Lt 代表程序总长度，Ln 和 Lr 分别为新编代码和复用代码的长度。令 Et、En 和 Er 分别代表开发该程序的总工作量、新编程序的工作量和复用已有软件构件的工作量，Cn 和 Cr 分别代表开发新代码和复用已有软件构件的生产率。

供选择的答案：

A ~ E. ① 知识复用 ② 分析结果复用 ③ 代码复用

④ 方法与标准复用 ⑤ 设计结果复用 ⑥ 软件成分复用

F~H. →① $\frac{L_t}{L_r}$ → → ② $\frac{L_r}{L_t}$ → → ③ $\frac{C_d}{1-(1-C_d/C_r)R}$

④ $\frac{L_t}{E_t}$ → ⑤ $\frac{L_d}{E_d}$ → ⑥ $\frac{C_d}{(1-C_d/C_r)R}$

⑦ $\frac{E_d}{(1-E_d/E_r)L_t}$

答案：A. ①, B. ④, C. ③, D. ⑤, E. ②, F. ④, G. ②, H. ③。

分析：广义来讲，软件复用可分为以下三个层次：

(1) 知识复用（如软件工程知识的复用）；

(2) 方法和标准的复用（如面向对象方法或国家制定的软件开发规范的复用）；

(3) 软件成分的复用。软件成分的复用又可进一步划分为三个级别：① 代码复用（源代码剪贴、源代码包含、继承）；② 设计结果复用；③ 分析结果复用（这是一种更高级别的复用，即复用某个系统的分析模型）。

为实现软件复用需要付出额外代价，如投资、时间和可复用构件库。

即使不考虑上述的额外代价，软件复用也不是必然能提高软件生产率。为研究软件复用程度与软件生产率的关系，我们首先引入下列两个量： 生产率 P = 程序总长度 / 开发该程序所用人时数

复用率 R = 复用代码长度 / 程序总长度

令 Lt 为程序总长度(目标代码条数)，Ln 和 Lr 分别为新编代码和复用代码的长度(都用目标代码条数来度量)。再令 Et、En 和 Er 分别代表开发该程序的总工作量、新编程序的工作量和复用已有软件构件的工作量，则有以下等式成立：

Lt = Ln + Lr, Et = En + Er, P = Lt / Et, R = Lr / Lt.

此外，用符号 Cn 和 Cr 分别代表开发新代码和复用已有软件构件的生产率，则有：

Cn = Ln / En, Cr = Lr / Er.

从上述 6 个等式出发，可以推导出下列的生产率与复用率之间的关系：

$$P = \frac{C_e}{1-(1-C_e/C_r)R}$$

从上式可知，复用率 R 越高，生产率不一定就越高。只有当软件开发人员使用已有的软件构件构造应用系统时，其工作效率比重新从底层编写程序的效率高时，复用率的提高才会导致生产率提高。可见，通过软件复用来提高软件生产率，并不是一件轻而易举的事情。构件的实用程度和使用方便程度，以及软件人员的素质、开发环境等因素，都直接影响软件复用的效果。

四、习题

【8-1】软件复用的含义是什么？软件复用的范围有哪些方面？

【8-2】简述软件复用的大致过程。说明在此过程中每个步骤需采用的关键步骤。

【8-3】回答下列问题。

(1) 比较横向复用和纵向复用的异同及优劣。

(2) 假定有一个结构形如图 8.3 所示的构件库，基于该构件库的软件复用属于横向复用还是纵向复用？为什么？

【8-4】当今大多数软件复用过程中，都涉及创建过程的一个重要活动，即界定潜在的可复用的资源。在这个活动中需要一整套的界定方法，并需要一个能确保可复用资源被复用的体系结构。这个活动就叫做(A)。而应用软件的开发过程或复用过程则称为(B)。系统地软件复用的实质是：(C)先投资，即界定并仔细地创建出可复用的资源，从而可以使得(D)能够又快又省地开发应用软件。

供选择的答案：

A, B. ① 应用系统工程 ② 软件工程 ③ 论域工程 ④ 需求工程

C, D. ① 用户 ② 创建者 ③ 复用者 ④ 投资者 ⑤ 管理者

【8-5】所谓一个“构件”，可以是一个类型、类或其它的工作成品。对于构件，应当按可复用的要求(A)、(B)、打包、编写文档。构件是(C)，并具有相当稳定的公开的(D)。这里的构件是基于(E)技术的。(E)技术中的封装、多态等特性，可简化构件的开发工作。而(E)技术中的(F)机制则有使开发简化、维护复杂的两面性。

供选择的答案：

A, B. ① 实现 ② 运行 ③ 设计 ④ 计划

C, D. ① 外联的 ② 内聚的 ③ 公开的 ④ 私有的

⑤ 外设 ⑥ 接口 ⑦ 输出

E, F. ① 面向数据 ② 面向对象 ③ 封装 ④ 继承 ⑤ 复用

【8-6】软件开发人员必须用各种抽象化手段来开发代码级构件。典型的抽象方法有 3 种。(A)、(B)、(C)。(A)使构件的(D)仅由接口说明确定，而把实现细节对构件的使用者隐藏起来；(B)在(A)的基础上进一步隐藏除接口参数外的所有数据。面向对象程序设计语言中的“类”是一种典型的基于(B)的构件。(C)则在(B)的基础上进一步实现在同一构件上并发执行的多个(E)的无关性。构件提供端口，以便为多个(F)同时访问构件中的资源进行(G)控制。

供选择的答案：

A ~ B. ① 数据抽象 ② 需求抽象 ③ 设计抽象 ④ 过程抽象

⑤ 功能抽象 ⑥ 实例抽象

D ~ G. ① 异步 ② 程序 ③ 线程 ④ 用户

⑤ 功能 ⑥ 同步

【8-7】所谓分层式体系结构是按层组织的软件的一种软件体系结构，其中，每一层的软件建立在低一层的软件层上。低层软件比高层软件更具(A)。右图是一种 4 层次的分层体系结构，请选择 B、C、D、E。

供选择的答案：

A. ① 高效性 ② 通用性

③ 可靠性 ④ 保密性

B ~ E. ① 系统软件 ② 特定业务

③ 中间件 ④ 应用软件 ⑤ 软件平台

【8-8】回答下列问题。

(1) 简述关键词分类法、多面分类法和超文本分类法等组织方法的概要。

(2) 简述相应的检索方法。

【8-9】可以有很多方法来描述可复用的构件，其中最理想的是由 Tracz 提出的 3C 模型。3C 是指(A)、(B)和(C)。(A)描述构件做什么。为此，需要完全地描述构件的(D)和表示(E)。(B)描述构件的(F)如何实现。一般来讲，(B)是对外来用户(G)的信息，是只有试图修改或测试该构件的人才需要了解的信息。(C)将通过定义概念的、操作的和实现的特征，使用户能够找到合适的构件以满足应用的需求。

供选择的答案：

A ~ C. ① 包容(container) ② 上下文(context) ③ 代价(cost)

④ 控制(control) ⑤ 概念(concept) ⑥ 内容(content)

D ~ C. ① 概念 ② 接口 ③ 公开 ④ 服务

⑤ 语义 ⑥ 语法 ⑦ 隐藏 ⑧ 结构

【8-10】试简述基于软件复用的软件项目管理机构的组织和工作职责。

五、习题解答

【8-1】软件复用就是指利用某些早先开发的对建立新软件系统有用的软件元素来生产新系统。软件复用的范围有以下十个方面：

① 项目计划：软件项目计划的基本结构和许多内容都可以跨项目复用。

② 成本估计：由于不同项目中常包含类似的功能，所以有可能在极少修改或不修改的情况下，复用对该功能的成本估计。

③ 体系结构：即便应用论域千差万别，但程序和数据体系结构大同小异。因此，可以创建一组类属的体系结构模板，将这些模板作为可复用的涉及框架。

④ 需求模型和规格说明：类和对象的模型和规格说明显然可以复用。此外，用传统软件工程方法开发的分析模型也可以复用。

⑤ 设计：用传统方法开发的体系结构、数据、接口和过程化设计，以及用面向对象方法开发的系统和对象的设计都可以复用。

⑥ 源程序代码：验证过的程序构件可以拿来复用的。

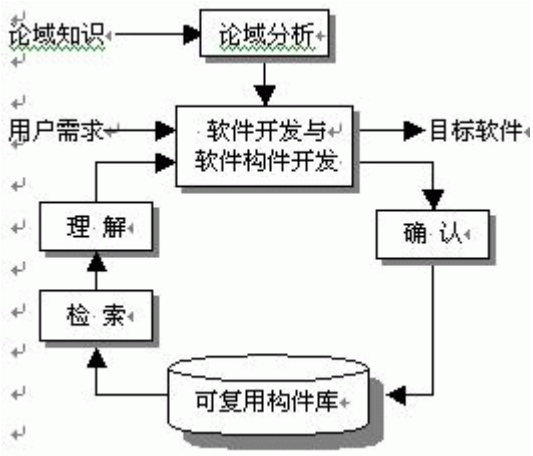
⑦ 用户文档和技术文档：即便特定的应用不同，但也经常有可能复用用户文档和技术文档中的大部分内容。

⑧ 用户界面：这可能是最广泛被复用的软件元素。如经常复用 GUI 的软件构件。

⑨ 数据结构：经常被复用的数据结构包括：内部表、列表和记录结构，以及文件和完整的数据库。

⑩ 测试用例：只要将设计或代码构件定义成可复用构件，相关的测试用例就应当成为这些构件的“从属品”。

【8-2】软件复用过程如图所示。



主要关键点在于论域分析：根据应用论域的特征和相似性，预测构件的可复用性。一旦根据论域分析确认了构件的可复用价值，即可进行构件的开发，并对具有可复用价值的构件做一般化处理，使它们能够适应新的类似的应用论域。然后将软件构件和它们的文档存入可复用构件库，成为可供未来开发项目使用的可复用资源。这些步骤构成构件的构造活动。 随着可复用构件的不断丰富，可复用构件库的规模会不断扩大，必须考虑如何组织库的结构以保持较高的检索效率。可供选择的软件构件从库中被检索出来之后，用户还必须理解它的功能或行为，以判定它是否真正适合于当前的应用。必要时，可考虑对某个与期望的功能或行为匹配程度最好的可复用构件进行少量修改，甚至可将修改后的构件再加到可复用构件库中。 软件复用的过程可借助计算机的帮助。支持软件复用的 CASE 工具的主要工作是：用某种组织形式实现可复用构件的存储，提供友好的人机界面，帮助用户浏览、检索和修改可复用构件库，对用户感兴趣的软件构件进行解释。事实上，几乎所有的软件复用活动都是在 CASE 工具的帮助下进行的。下面列出软件复用各个步骤中的关键技术。

论域分析	构件开发	构件库的组织与检索	构件合成
知识获取	提高构件独立性	构件库的组织与存储	构件修改
对象及其操作标识	参数化	浏览	构件实例化
抽象与关联	抽象与特殊化	检索	修改后构件正确性验证
对象分类	构件的扩充	结构化查询	将构件合成到目标软件系统
论域语言处理	测试与认证	基于语义的检索	
	形式验证		
	构件质量评估		
	构件分类		

【8-3】(1) 软件复用可以区分为横向复用和纵向复用。横向复用是复用不同应用论域中的软件元素，例如数据结构、排序算法、人机界面构件等。标准函数库是一种典型的原始的横向复用机制。纵向复用是在一类具有较多公共性的应用论域之间复用软件构件。 横向复用是在几个截然不同的应用论域之间进行软件复用，可复用的资源有限，潜力不大。纵向复用的应用范围较大，从系统软件到特定论域的软件及应用软件，还有软件工程过程方面，许多计算机软件厂商开发了大量可复用的构件或软件包，在软件工程实践中取得显著的效益，因此受到广泛关注。

(2) 图 8.3 是关键词分类结构的示例，它是横向复用的典型例子，用户界面所涉及的技术可用于各个不同应用论域的软件中。

【8-4】A. ③, B. ①, C. ②, D. ③

当今大多数软件复用过程中，都涉及创建过程的一个重要活动，即界定潜在的可复用的资源。在这个活动中需要一整套的界定方法，并需要一个能确保可复用资源被复用的体系结构。这个活动就叫做论域工程。而应用软件的开发过程或复用过程则称为应用系统工程。系统地软件复用的实质是：创建者先投资，即界定并仔细地创建出可复用的资源，从而可以使得复用者能够又快又省地开发应用软件。

【8-5】A. ③, B. ①, C. ②, D. ⑥, E. ②, F. ④。

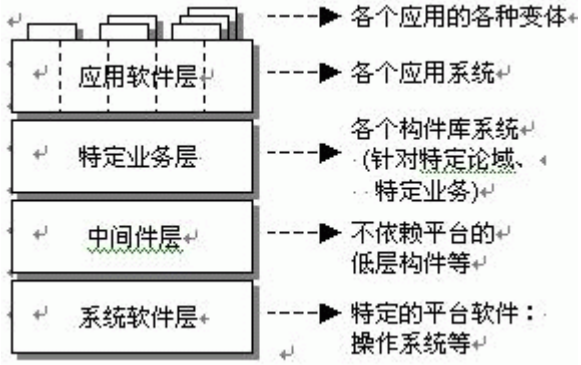
所谓一个构件，可以是一个类型、类或其它的工作成品。对于构件应当按可复用的要求设计、实现、打包、编写文档。构件是内聚性很强的，并具有相当稳定的公开的接口。这里的构件是基于面向对象技术的。面向对象技术中的封装、多态等特性，可简化构件的开发工作。而面向对象技术中的继承机制则有两面性，一方面可以简化开发工作，另一方面使构件的维护变得很复杂。

【8-6】A. ⑤, B. ①, C. ④, D. ⑤, E. ③, F. ④, G. ⑥。

软件开发人员必须用各种抽象化手段来开发代码级构件。典型的抽象方法有 3 种。功能抽象、数据抽象和过程抽象。功能抽象要求构件的功能仅由接口说明中的输入－关系确定，把构件的实现细节对构件的使用者隐藏起来。数据抽象则在功能抽象的基础上进一步隐藏除接口参数外的所有数据。构件的功能或行为可以由输入参数和构件自身记忆的内部状态决定。面向对象程序设计语言中的“类”是一种典型的基于数据抽象的构件。而过程抽象在数据抽象的基础上进一步实现在同一构件上并发执行的多个线程的无关性。构件提供端口，以便为多个用户同时访问软件构件中的资源进行同步控制。访问请求首先进入等待队列，构件就绪后，从队列中取出请求逐个执行。使用构件的多个线程之间通过全局共享数据或消息传递进行信息交换。构件的设计必须保证构件接口元素的功能和行为在语义上的正确性，不管各线程操作序列之间的相对时序如何变化。

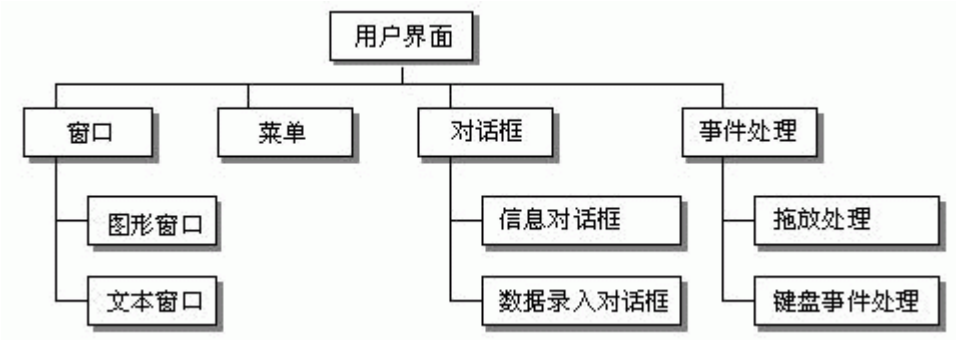
【8-7】A. ②, B. ④, C. ②, D. ③, E. ①。

在分层式体系结构中，位于同一层上的各个软件系统或子系统具有同等的通用性。每一层的软件建立在低一层的软件层上。低层软件比高层软件更具通用性。最高层是应用软件层，可包容许多应用系统。次高层是特定业务层，包括针对不同业务类型的许多构件库。中间件层位于特定业务层下面，它为其上层提供各种实用软件类，最低层是系统软件层，它包含许多基础设施软件，如操作系统、网络接口软件等。这种 4 层次的分层体系结构见图。



【8-8】(1) 软件构件的分类方法和相应的构件库结构对构件的检索和理解有极为深刻的影响，就是说，可复用构件库的组织应当便于构件的存储和检索。

① 关键词分类法。此方法的基本思想是：根据论域分析的结果，将应用论域(族)的概念按照从抽象到具体的顺序，逐步分解树形结构或有向无回路图结构。每个概念用一个描述性的关键词表示。不可再分解的原子层的包含隶属于它的某些软件构件。下图给出了可复用构件库的关键词分类结构，它支持图形用户界面设计。



当加入新的构件时，库管理员应对构件的功能或行为进行分析，浏览上述关键词分类结构，将构件置于最合适的原子层关键词下。如果无法找到构件应从属的关键词，可以扩充关键词分类结构，引进新的关键词。但必须保证新关键词有相应的论域分析的结果作为支持。

② 多面分类法。此方法由 3 部分构成：多面分类机制、同义词库和概念距离图。

ｉ）多面分类机制：分析论域范围并定义若干用于描述一个构件特征的“面”，每个“面”包含若干“概念”，它们表述构件在“面”上的基本特征。这些特征

根据它们的重要性排队。“面”可以描述构件执行的功能、被操作的数据、构件应用的上下文以及任何其它特征。描述某一构件的“面”的集合称为面描述子。

ii）同义词库：意义相同或相近的若干词汇组成同义词库。所有词汇按照隶属于“面”的“概念”分组，在任一时刻点，每个“概念”可用组内的某一同义词汇作为标识载体。

iii)概念距离图:用于度量每个“面”中“概念”的相似性程度。属于每个“面”的一般化概念与其中的两个或多个“概念”以加权边相连接,两个“概念”的相似性由它们之间的最短加权路径上的加权距离确定，附加于边上的权值体现了“概念”之间的差异程度。

采用多面分类法进行可复用构件库的组织，必须在存储软件构件的同时，表示并存储多面分类机制、同义词库和概念距离图。多面分类法的所有语法构件(“面”、“概念”、同义词、一般化概念、差异性权值)均取材于论域分析的结果。当需要在可复用构件库中加入新的构件时，库管理源必须对构件的功能、行为进行深入分析，利用现有的多面分类结构确定构件的描述子，对每个“面”选取合适的“概念”作为特征描述。必要时可以考虑增加新的“概念”，此时必须根据新“概念”完善同义词库和概念距离图。

③ 超文本组织法。此方法的基本思想是：所有构件都必须辅以详细的功能或行为说明文档，说明中出现的概念和构件以网状链接方式互相连接。检索者在阅读文档的过程中可任意跳转到包含相关概念或软件构件的文档中去。全文检索系统将用户给出的关键词与说明文档中的文字进行匹配，实现软件构件的浏览式检索。 超文本方法以结点为基本单位，链作为结点之间的联想式关联。一般地，结点是一个信息块。对于可复用构件库而言，结点可以是论域的概念、功能或行为名称、构件名称等，在图形用户界面上，结点可以是字符串，也可以是图像、声音、动画等。

(2) 可复用构件库的检索方法与库的组织方式密切相关。

① 基于关键词的检索：这种检索方法的基本思想是：系统(CASE 工具)在图形用户界面上将可复用构件库的关键词树直观地展示给用户,用户通过在树上的逐级浏览寻找需要的关键词并提取相应的构件。当然,用户也可以直接给出关键词(其中可含有通配符)，由系统自动地给出合适的候选构件清单。这种方法的优点是简单，易于实现。缺点是对库的浏览容易使用户迷失方向。

② 多面检索：这种检索方法基于多面分类法。步骤如下：

§ 构造查询：用户提供待查构件在每个“面”上的特征，生成构件描述子。此时，用户可以从可复用构件库已有的“概念”中挑选，也可以将某些特征值置为空。系统在库中检索时将忽略特征值为空的“面”。在构造查询的过程中，可以利用同义词库和概念距离图来帮助用户正确选择特征值。

§ 检索构件：实现多面分类法的 CASE 工具利用同义词库和概念距离图，在可复用构件库中寻找相同的或相近的构件描述子及相应的构件。

§ 对构件进行排序：按照相似程度对被检索出来的构件进行排序，还可以按照与复用有关的度量信息(如构件的复杂性、可复用性、成功复用的次数等)进行排序。

这种方法的优点是易于实现相似构件的查找，但用户构造查询比较麻烦。

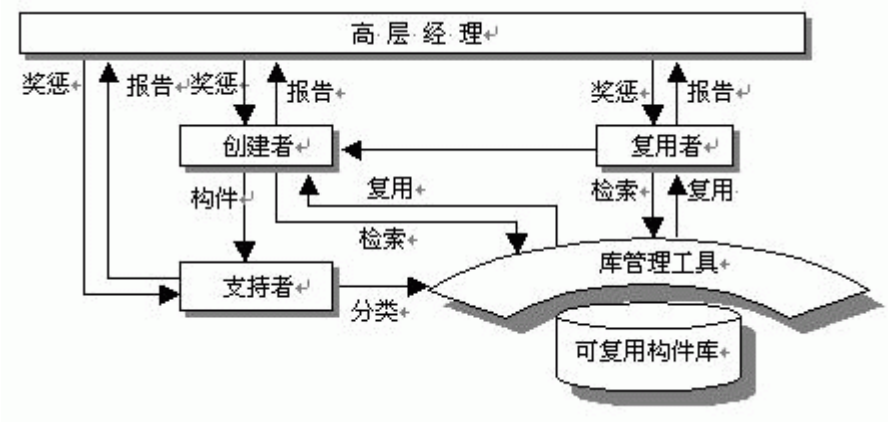
③ 超文本检索：这种检索方法的步骤是：用户首先给出一个或若干个关键词，系统在构件的说明文档中做精确的或模糊的语法匹配。匹配成功后，向用户提供相应的构件说明。这些构件说明是含有许多超文本结点的正文。用户在阅读这些正文时可实现多个构件说明文档之间的自由跳转，最终选择合适的构件。为了避免用户在跳转过程中迷失方向，系统可以通过图形用户界面显示浏览历史图，允许将特定画面定义为有名“书签”并可随时跳转到“书签”。此外，还可以帮助用户逆着跳转路径逐步返回。 这种方法的优点是用户界面友好，但在某些情况下用户难以在超文本浏览过程中正确选择软件构件。

④ 其它检索方法：上述检索方法都是基于语法匹配，要求用户对可复用构件库中的出现的众多词汇有较全面的把握，较精确的理解。理论上，理想的检索方法是语义匹配：可复用构件库的用户以形式化的手段描述所需要的构件的功能或行为的语义，系统通过定理证明或基于知识的推理过程寻找语义上等价的或相近的构件。但这种基于语义的检索方法涉及许多人工智能的难题，目前难于支持大型构件库的工程实现。

【8-9】A. ⑤, B. ⑥, C. ②, D. ②, E. ⑤, F. ①, G. ⑦。

可以有很多方法来描述可复用的构件，其中最理想的是由 Tracz 提出的 3C 模型。3C 是指概念(concept)、内容(content)和上下文(context)。概念描述软件构件做什么。为此，需要完全地描述构件的接口和表示语义(表示在前置条件和后置条件的上下文中)。概念将传达构件的意图。内容描述软件构件的概念如何实现。一般来讲，内容是对外来用户隐蔽的信息，是只有试图修改或测试该构件的人才需要了解的信息。上下文将可复用的构件安置到它的应用论域中。就是说，上下文将通过定义概念的、操作的和实现的特征，使软件人员能够找到合适的构件以满足应用的需求。

【8-10】基于复用的软件项目组织与传统的软件开发项目组织不同，它必须有两个职能，并由两个部门分别承担这两个职能。一个职能是创建，相应的部门是创建者或论域工程部门；另一个职能是复用，相应的部门是复用者或应用工程部门。而具有复用经验的单位，往往还需要第三个职能，即支持，相应的部门是支持者或支持部门。创建、复用和支持这三个平行的部门之上，还有一个高层经理，他关注的是总目标。



① 复用者负责传统的软件开发任务。同时，在各开发阶段，软件人员可查询构件库,从中检索可复用构件,进行适应性修改后将其合成到当前的目标软件中来。此外，对于同类(或同族)应用的首次开发，复用者还需在做需求分析之前进行论域分析，以便为系统地构造软件构件提供依据。

② 创建者负责从复用者手中接受构件的设计规格说明，进行构件的设计、实现和质量控制，重点考虑因素是构件的可复用性。在构件库的积累初期，开发可复用构件的工作量较大，需要设立专门的软件构件开发组。但随着构件库的不断丰富，可以考虑将构件开发组并入系统开发组。

③ 高层经理负责管理、组织和协调各类软件复用活动，并处理所有有关复用的事务性工作。所有下属的创建、复用和支持部门都应定期向他报告各自复用活动的当前状况，高层经理根据这些状况制定或调整复用计划，实施奖励，以调动开发人员的复用积极性。此外，他还应根据整体复用情况，向项目管理人员提供资源分配、进度安排等方面的决策建议。

④ 复用支持者负责可复用构件的资格确认、质量保证、分类和存储。在可复用构件库尚未形成的初期,支持者的任务还包括开发可复用构件库和其它配套的软件复用 CASE 工具。此后，还可将部分力量投入应用论域中某些通用构件的开发。

第九章 软件管理

一、复习要求

1. 了解软件过程的概念、软件过程框架和软件过程模型。

2. 了解软件项目管理的过程。

3. 了解软件度量的种类，面向规模和面向功能的度量以及质量度量的种类。

4. 掌握 LOC 估算和 FP 估算的方法，分解技术和工作量估算方法。

5. 了解软件成本估算的概念，掌握 COCOMO 成本估算方法。

6. 了解软件成本－效益估计方法。

7. 了解风险分析的步骤，风险的种类、风险项目和风险构成。

8. 了解软件进度安排方法及图形工具。

9. 了解软件项目划分的方式，项目组织的模式，人员配备的原则和条件。

二、内容提要

1. 软件过程

(1) 软件过程的概念

软件工程是一种层次化的技术，如图 9.1 所示。软件工程的过程层是将结合在一起的凝聚力量，使得计算机软件能够及时、合理地被开发出来。软件过程定义了一组关键过程域(KPAs)，它们构成软件项目管理的基础，并规定了技术方法的采用、工程产品(模型、文档、数据、报告、表格等)的产生、里程碑的建立、质量的管理以及适当的变更控制。



图 9.1 软件工程层次

软件过程是软件生存期中的一系列相关软件工程活动的集合。每一个软件过程又是由一组工作任务、项目里程碑、软件工程产品和交付物以及质量保证(SQA)点等组成。一个软件过程可以用图 9.2 的形式来表示。首先建立一个公共过程框架，其中定义了少量可适用于所有软件项目的框架活动，而不考虑它们的规模和复杂性。再给出各个框架活动的任务集合，使得框架活动能够适合于项目的特点和项目组的需求。最后是保护伞活动，如软件质量保证、软件配置管理以及测量等，它们独立于任何一个框架活动并将贯穿于整个过程。

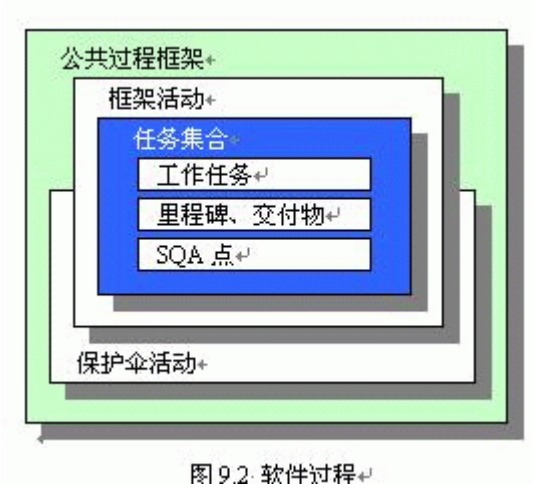


图 9.2 软件过程

(2) 软件过程模型

软件工程过程模型的选择基于项目和应用的特点、采用的方法和工具、要求的控制和需交付的产品。L.B.S.Raccoon 使用了分级几何表示，用以讨论软件工程过程的本质。

所有的软件开发都可以看成是一个问题循环解决过程，如图 9.3 所示。其中包括 4 个截然不同的阶段：状态捕获、问题定义、技术开发和方案综合。状态捕获表示了事物的当前状态；问题定义标识了需要解决的特定问题；技术开发利用某些技术来解决问题；方案综合导出最终的结果(如文档、程序、数据、新的事务功能、新的产品)。

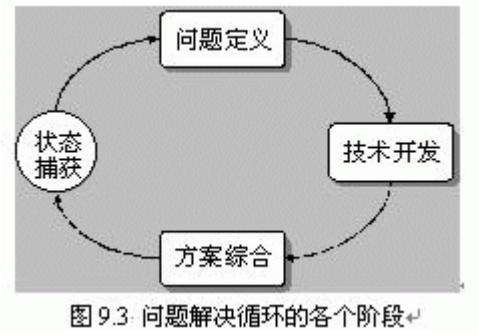


图 9.3 问题解决循环的各个阶段

以上的问题循环解决过程可以用于软件工程的不同开发级别上。它可用于考虑整个应用系统的宏观级，也可用于建造程序构件的中间级，甚至还可用于源代码行级。因此，可以用分级几何表示来给出过程的理想化的视图。首先定义一个分级几何表示的模式，然后相继地在更小的规模上递归地应用分级几何表示：模式中嵌套模式。在图 9.4 中，问题循环解决过程的每一个阶段又包含一个同样的问题循环解决过程，该循环中每一个步骤中还可以再包含另一个问题循环解决过程。这样一直继续下去，直到某个合理的边界为止。对于软件来说，就是源代码行。

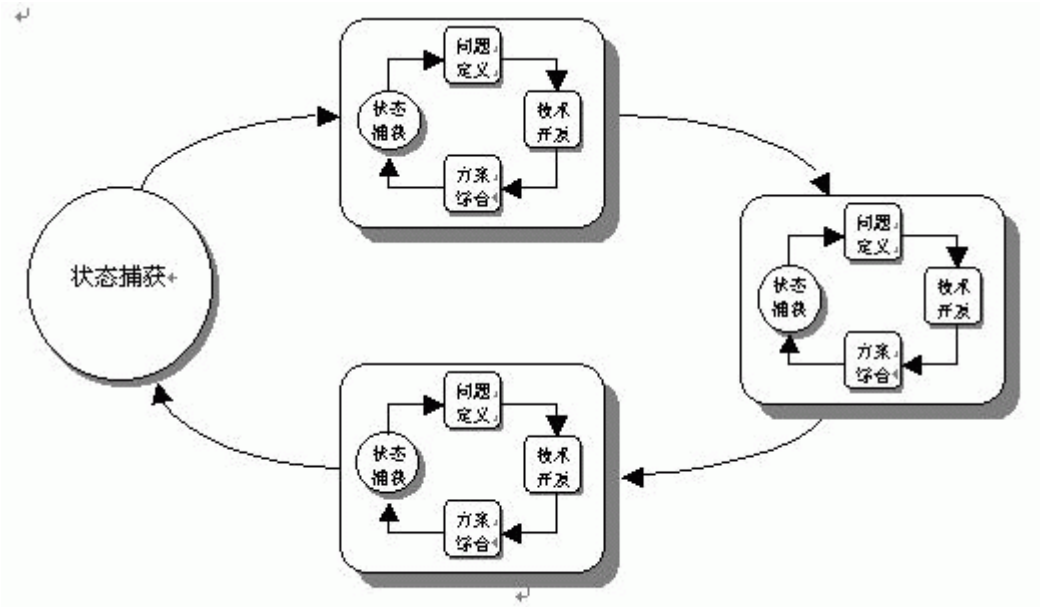


图 9.4 问题循环解决过程中阶段嵌套阶段

实际上，想要如图 9.4 那样清楚地划分这些活动是很困难的，因为在阶段内部常常会出现一些交叉的任务，它们还可能会跨越阶段。不过，这种简化的视图表达了一个重要的思想：不管软件项目选择了什么样的过程模型，但所有阶段，包括状态捕获、问题定义、技术开发、方案综合，在某个细节级别上都同时存在。由于给出了如图 9.4 所示的递归的性质，上述的 4 阶段论不但可用于整个应用的分析，而且同样地可用于某一代码段的生成。

(3) 过程建造技术

为使得软件过程模型适合于软件项目组的使用，需要开发一些过程技术工具，以帮助软件开发组织分析它们当前的过程，组织工作任务，控制和监控进度，管理技术质量。

使用过程技术工具，可以建造一个自动模型，模型包含前面提到的公共过程框架、任务集合及保护伞活动。该模型一般表示成一个网络，对其加以分析，就能够确定典型的工作流程，考察可能导致减少开发时间、降低开发成本的可选的过程结构。

一旦创建了一个可接受的过程，就可以使用其它过程技术工具来分配、监视、甚至控制在软件过程模型中定义的所有软件工程任务。软件项目组的每一个成员都可以使用这样的工具来开发检查表，列出所有将要执行的工作任务、将要产生的工作产品和将要实施的软件质量保证活动。过程技术工具还可用于协调适合某一特定工作任务的其它 CASE 工具的使用。

2、软件项目管理过程

软件项目管理包括进度管理、成本管理、质量管理、人员管理、资源管理、标准化管理。管理的对象是进度、系统规模及工作量估算、经费、组织机构和人员、风险、质量、作业和环境配置等。软件项目管理所涉及的范围覆盖了整个软件生存期。

为使软件项目开发获得成功，一个关键问题是必须对软件开发项目的工作范围、可能遇到的风险、需要的资源(人、硬 / 软件)、要实现的任务、经历的里程碑、花费工作量(成本)，以及进度的安排等等做到心中有数。而软件项目管理可以提供这些信息。通常，这种管理在技术工作开始之前就应开始，而在软件从概念到实现的过程中继续进行，并且只有当软件开发工作最后结束时才终止。

(1) 启动一个软件项目

在制定软件项目计划之前，必须先明确项目的目标和范围、考虑候选的解决方案、标明技术和管理上的要求。有了这些信息，才能确定合理、精确的成本估算，实际可行的任务分解以及可管理的进度安排。

项目的目标标明了软件项目的目的但不涉及如何去达到这些目的。范围标明了软件要实现的基本功能，并尽量以定量的方式界定这些功能。候选的解决方案虽然涉及方案细节不多，但有了方案，管理人员和技术人员就能够据此选择一种“好的”方法，给出诸如交付期限、预算、个人能力、技术界面及其它许多因素所构成的限制。

(2) 制定项目计划

制定计划的任务包括：

§ 估算所需要的人力（通常以人月为单位）、项目持续时间（以年份或月份为单位）、成本（以元为单位）。

§ 作出进度安排，分配资源，建立项目组织及任用人员(包括人员的地位、作用、职责、规章制度等)，根据规模和工作量估算分配任务。

§ 进行风险分析，包括风险识别、风险估计、风险优化、风险驾驭策略、风险解决和风险监督。这些步骤贯穿在软件工程过程中。

§ 制定质量管理指标：如何识别定义好的任务？管理人员对结束时间如何掌握，并如何识别和监控关键路径以确保结束？对进展如何度量？以及如何建立分隔任务的里程碑。

§ 编制预算和成本。

§ 准备环境和基础设施等。

(3) 计划的追踪和控制

一旦建立了进度安排，就可以开始着手追踪和控制活动。由项目管理人员负责在过程执行时监督过程的实施，提供过程进展的内部报告，并按合同规定向需方提供外部报告。对于在进度安排中标明的每一个任务，如果任务实际完成日期滞后于进度安排，则管理人员可以使用一种自动的项目进度安排工具来确定在项目的中间里程碑上进度误期所造成的影响。可对资源重新定向，对任务重新安排，或者(做为最坏的结果)可以修改交付日期以调整已经暴露的问题。用这种方式可以较好地控制软件的开发。

(4) 评审和评价计划的完成程度

项目管理人员应对计划完成程度进行评审，对项目进行评价。并对计划和项目进行检查，使之在变更或完成后保持完整性和一致性。

(5) 编写管理文档

项目管理人员根据合同确定软件开发过程是否完成。如果完成，应从完整性方面检查项目完成的结果和记录，并把这些结果和记录编写成文档并存档。

3、软件生产率和质量的度量

(1) 软件度量

对软件进行度量，是为了表明软件产品的质量，弄清软件开发人员的生产率，给出使用了新的软件工程方法和工具所得到的(在生产率和质量两方面)的效益，建立项目估算的“基线”，帮助调整对新的工具和附加培训的要求。

软件度量分为两类：

§ 直接度量：软件工程过程的直接度量包括所投入的成本和工作量。软件产品的直接度量包括产生的代码行数(LOC)、执行速度、存储量大小、在某种时间周期中所报告的差错数。

§ 间接度量：产品的间接度量则包括功能性、复杂性、效率、可靠性、可维护性和许多其它的质量特性。

只要事先建立特定的度量规程，很容易做到直接度量开发软件所需要的成本和工作量、产生的代码行数等。但是，软件的功能性、效率、可维护性等质量特性却很难用直接度量判明，只有通过间接度量才能推断。 我们可进一步将软件度量如图 9.5 所示那样分类。软件生产率度量主要关注软件工程过程的结果；软件质量度量则指明了软件适应明确和不明确的用户要求(软件使用合理性)到什么程度；技术度量主要关注软件的一些特性(如逻辑复杂性、模块化程度)而不是软件开发的全过程。

从图 9.5 中还可以看到另一种分类方法：面向规模的度量用于收集与直接度量有关的软件工程输出的信息和质量信息。面向功能的度量提供直接度量的尺度。面

向人的度量则收集有关人们开发软件所用方式的信息和人们理解有关工具和方法的效率的信息。

(2) 面向规模的度量

面向规模的度量是对软件和软件开发过程的直接度量。首先需要建立一个如表 9.1 所示的面向规模的数据表格，记录过去几年完成的每一个软件项目和关于这些项目的相应面向规模的数据。

表 9.1 面向规模的度量

项目	工作量(人月)	元(千)	规模(KLOC)	文档页数	错误数	开发人数
aaa-01	24	168	12.1	365	29	3
ccc-04	62	440	27.2	1224	86	5
fff-03	43	314	17.5	1050	64	6
…	…	…	…	…	…	…

对于每一个项目，可以根据表格中列出的基本数据进行一些简单的面向规模的生产率和质量的度量。例如，可以根据表 9.1 对所有的项目计算出平均值：

生产率 = KLOC / PM（人月）
成本 = 元 / LOC

质量 = 错误数 / KLOC
文档 = 文档页数 / KLOC

(3) 面向功能的度量

面向功能的软件度量是对软件和软件开发过程的间接度量。面向功能度量的关注点在于程序的“功能性”和“实用性”，而不是对 LOC 计数。一种典型的生产率度量法叫做功能点度量，该方法利用软件信息域中的一些计数度量和软件复杂性估计的经验关系式而导出功能点 FPs（Function Points）。

功能点通过填写表 9.2 所示的表格来计算。首先确定五个信息域的特征，并在表格中相应位置给出计数。信息域的值以如下方式定义：

§ 用户输入数：各个用户输入是面向不同应用的输入数据，对它们都要进行计数。输入数据应有别于查询数据，它们应分别计数。

§ 用户输出数：各个用户输出是为用户提供的面向应用的输出信息，它们均应计数。这里的输出是指报告，屏幕信息，错误信息等，在报告中的各数据项不应再分别计数。

§ 用户查询数：查询是一种联机输入，它导致软件以联机输出的方式生成某种即时的响应。每一个不同的查询都要计数。 § 文件数：每一个逻辑主文件都应计数。这里的逻辑主文件，是指逻辑上的一组数据，它们可以是一个大的数据库的一部分，也可以是一个单独的文件

§ 外部接口数：对所有被用来将信息传送到另一个系统中的机器可读写的接口(即磁带或磁盘上的数据文件)均应计数。

表 9.2 功能点度量的计算

信息域参数	计数		加权因数				加权计数
			简单	中间	复杂		
用户输入数	□□□	×	3	4	6	=	□□□
用户输出数	□□□	×	4	5	7	=	□□□
用户查询数	□□□	×	3	4	6	=	□□□
文 件 数	□□□	×	7	10	15	=	□□□
外部接口数	□□□	×	5	7	10	=	□□□
总 计 数							□□□

一旦收集到上述数据，就可以计算出与每一个计数相关的复杂性值。使用功能点方法的机构要自行拟定一些准则以确定一个特定项是简单的、平均的还是复杂的。

计算功能点，使用如下的关系式：

FP ＝ 总计数×〔0.65＋0.01×SUM（Fi）〕 （9.1）

其中，总计数是由表 9.2 所得到的所有加权计数项的和；Fi（i = 1 到 14）是复杂性校正值，它们应通过逐一回答表 9.3 所提问题来确定。SUM（Fi）是求和函数。上述等式中的常数和应用于信息域计数的加权因数可经验地确定。

一旦计算出功能点，就可以仿照 LOC 的方式度量软件的生产率、质量和其它属性：

生产率 ＝ FP / PM（人月）
成本 ＝ 元 / FP

质量 ＝ 错误数 / FP
文档 ＝ 文档页数 / FP

表 9.3 计算功能点的校正值

评定每个校正因素的尺度是 0－5						
	0	1	2	3	4	5
	没有影响	偶然的	适中的	普通的	重要的	极重要的
Fi						
1	系统是否需要可靠的备份和恢复？					
2	是否需要数据通信？					
3	是否有分布式处理的功能？					
4	性能是否是关键？					
5	系统是否将运行在现有的高度实用化的操作环境中？					
6	系统是否要求联机数据项？					
7	联机数据项是否要求建立在多重窗口显示或操作上的输入事务？					
8	是否联机地更新主文件？					
9	输入、输出、文件、查询是否复杂？					
10	内部处理过程是否复杂？					
11	程序代码是否要设计成可复用的？					
12	设计中是否包含变换和安装？					
13	系统是否要设计成多种安装形式以安装在不同的机构中？					
14	应用系统是否要设计成便于修改和易于用户使用？					

功能点度量是为了商用信息系统应用而设计的。Jones 将其扩充，使这种度量可以被用于系统和工程软件应用，称之为特征点 FPs（Feature Points）。特征点度量适合于算法复杂性高的应用。实时处理、过程控制、嵌入式软件应用的算法复杂性都偏高，适于特征点度量。

为了计算特征点，可以象上面描述的那样，对信息域值进行计数和加权。此外，需要对一个新的软件特征“算法”进行计数。可定义算法为“在一个特定计算机程序内所包含的一个有界的计算问题。”如矩阵求逆、二进位串转换为十进制数、处理一个中断等都是算法。计算特征点可使用如表 9.4 所示的表格。 对于每一个度量参数只使用一个权值，并且使用等式（9.1）来计算总的特征点值。

表 9.4 特征点度量的计算

信息域参数	计数		权值		加权计数
用户输入数	□□□	×	4	=	□□□
用户输出数	□□□	×	5	=	□□□
用户查询数	□□□	×	4	=	□□□
文 件 数	□□□	×	7	=	□□□
外部接口数	□□□	×	7	=	□□□
算 法	□□□	×	3	=	□□□
总 计 数				→	□□□

必须注意，特征点与功能点表示的是同一件事：由软件提供的“功能性”或“实用性”。事实上，对于传统的工程计算或信息系统应用，两种度量会得出相同的

FP 值。在较复杂的实时系统中，特征点计数常常比只用功能点确定的计数高出 20％到 35％。

(4) 软件质量的度量

质量度量贯穿于软件工程的全过程中以及软件交付用户使用之后。在软件交付之前得到的度量提供了一个定量的根据，以做出设计和测试质量好坏的判断。这一类度量包括程序复杂性、有效的模块性和总的程序规模。在软件交付之后的度量则把注意力集中于还未发现的差错数和系统的可维护性方面。特别要强调的是，软件质量的售后度量可向管理者和技术人员表明软件工程过程的有效性达到什么程度。

虽然已经有许多软件质量的度量方法，但事后度量使用得最广泛。它包括正确性、可维护性、完整性和可使用性。Gilb 提出了它们的定义和度量。

① 正确性：一个程序必须正确地运行，而且还要为它的用户提供某些输出。正确性要求软件执行所要求的功能。对于正确性，最一般的度量是每千代码行(KLOC)的差错数，其中将差错定义为已被证实是不符合需求的缺陷。差错在程序交付用户普遍使用后由程序的用户报告，按标准的时间周期(典型情况是 1 年)进行计数。

② 可维护性：包括当程序中发现错误时，要能够很容易地修正它；当程序的环境发生变化时，要能够很容易地适应之；当用户希望变更需求时，要能够很容易地增强它。还没有一种方法可以直接度量可维护性，因此必须采取间接度量。有一种简单的面向时间的度量，叫做平均变更等待时间 MTTC（Mean Time To Change)。 这个时间包括开始分析变更要求、设计合适的修改、实现变更并测试它、以及把这种变更发送给所有的用户。一般地，一个可维护的程序与那些不可维护的程序相比，应有较低的 MTTC（对于相同类型的变更)。

③ 完整性：这个属性度量一个系统抗拒对它的安全性攻击(事故的和人为的)的能力。软件的所有三个成分：程序、数据和文档都会遭到攻击。

为了度量完整性，需要定义两个附加的属性：危险性和安全性。危险性是特定类型的攻击将在一给定时间内发生的概率，它可以被估计或从经验数据中导出。安全性是排除特定类型攻击的概率，它也可以被估计或从经验数据中导出。一个系统的完整性可定义为：

完整性 ＝ Σ（1－危险性×（1－安全性））

其中，对每一个攻击的危险性和安全性都进行累加。

④ 可使用性：即用户友好性。如果一个程序不具有“用户友好性”，即使它所执行的功能很有价值，也常常会失败。可使用性力图量化“用户友好性”，并依据以下四个特征进行度量：

§ 为学习系统所需要的体力上的和智力上的技能；

§ 为达到适度有效使用系统所需要的时间；

§ 当软件被某些人适度有效地使用时所度量的在生产率方面的净增值；

§ 用户角度对系统的主观评价(可以通过问题调查表得到)。

4、软件项目的估算

在做软件估算时往往存在某些不确定性，这将使得软件项目管理人员无法正常进行管理。因为估算是所有其它项目计划活动的基石，且项目计划又为软件工程过程提供了工作方向，所以我们不能没有计划就开始着手开发，否则将会陷入盲目性。

(1) 估算

估算资源、成本和进度时需要经验、有用的历史信息、足够的定量数据和作定量度量的勇气。估算本身带有风险。增加风险的各种因素如图 9.6 所示。图中的轴线表示被估算项目的特征。

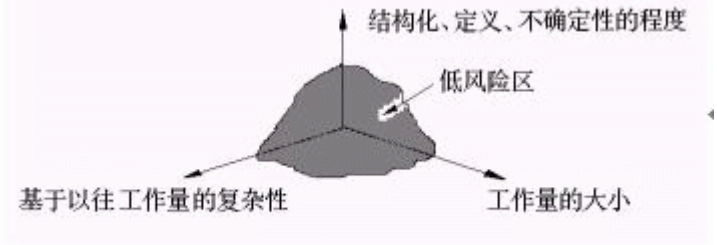


图 9.6 估算与风险

项目的复杂性对于增加软件估算的不确定性影响很大。复杂性越高，估算的风险就越高。但是，复杂性是相对度量，它与项目参加人员的经验有关。例如，一个实时系统的开发，对于过去仅做过批处理应用项目的软件开发组来说是非常复杂的，但对于一个过去开发过许多高速过程控制软件的软件小组来说可能就是很容易的了。此外，这种度量一般用在设计或编码级，而在软件计划时(设计和代码存在之前)使用就很困难。因此，可以在计划过程的早期建立其它较为主观的复杂性评估，如功能点复杂性校正因素。

项目的规模对于软件估算的精确性和功效影响也比较大。因为随着软件规模的扩大，软件元素之间的相互依赖、相互影响程度迅速增加，因而估算的一个重要方法——问题分解会变得更加困难。由此可知，项目的规模越大，开发工作量越大，估算的风险越高。 项目的结构化程度也影响项目估算的风险。所谓结构性是指

功能分解的简便性和处理信息的层次性。结构化程度的提高，进行精确估算的能力就能提高，而风险将减少。

历史信息的有效性也影响估算的风险。回顾过去，就能够仿效做过的事，且改进出现问题的地方。在对过去的项目进行综合的软件度量之后，就可以借用来比较准确地进行估算，安排进度以避免重走过去的弯路，而总的风险也减少了。

风险靠对不确定性程度定量地进行估算来度量，此外，如果对软件项目的作用范围还不十分清楚，或者用户的要求经常变更，都会导致对软件项目所需资源、成本、进度的估算频频变动，增加估算的风险。计划人员应当要求在软件系统的规格说明中给出完备的功能、性能、接口的定义。更重要的是，计划人员和用户都应认识到经常改变软件需求意味着在成本和进度上的不稳定性。

(2) 软件的范围

软件项目计划第一项活动就是确定软件的范围。应当从管理角度和技术角度出发，确定明确的和可理解的项目范围，明确地给出定量的数据(如同时使用该软件的用户数目,发送表格的长短,最大允许响应时间等),指明约束条件和限制(如存储容量)。此外还要叙述某些质量因素(如给出的算法是否容易理解、是否使用高级语言等)。

软件范围包括功能、性能、限制、接口和可靠性。在估算开始之前，应对软件的功能进行评价，并对其进行适当的细化以便提供更详细的细节。由于成本和进度的估算都与功能有关，因此常常采用某种程度的功能分解。性能的考虑包括处理和响应时间的需求。约束条件则标识外部硬件、可用存储或其它现有系统对软件的限制。

功能、性能和约束必须在一起进行评价。当性能限制不同时，为实现同样的功能，开发工作量可能相差一个数量级。功能、性能和约束是密切联系在一起的。 软件与其它系统元素是相互作用的。计划人员要考虑每个接口的性质和复杂性，以确定对开发资源、成本和进度的影响。接口的概念可解释为：

§ 运行软件的硬件(如处理机与外设)及间接受软件控制的设备(如机器、显示器)；

§ 必须与新软件链接的现有的软件(如数据库存取例程、子程序包、操作系统)；

§ 通过终端或其它输入 / 输出设备使用该软件的人；

§ 该软件运行前后的一系列操作过程。

对于每一种情况，都必须清楚地了解通过接口的信息转换。

软件范围最不明确的方面就是可靠性的讨论。软件可靠性的度量已经存在，但它们在项目的这个阶段难得用上。因此，可以按照软件的一般性质规定一些具体的要求以保证它的可靠性。

(3) 软件开发中的资源

软件项目计划的第二个任务是对完成该软件项目所需的资源进行估算。图 9.7 把软件开发所需的资源画成一个金字塔，在塔的底部有现成的用以支持软件开发的工具——硬件及软件工具，在塔的高层是最基本的资源——人。通常，对每一种资源，应说明 4 个特性：资源的描述，资源的有效性说明，资源在何时开始需要，使用资源的持续时间。最后两个特性统称为时间窗口。对每一个特定的时间窗口，在开始使用它之前就应说明它的有效性。

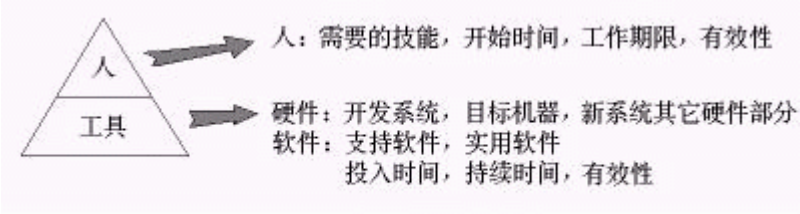


图 9.7 软件开发所需的资源

① 人力资源

在考虑各种软件开发资源时，人是最重要的资源。在安排开发活动时必须考虑人员的技术水平、专业、人数、以及在开发过程各阶段中对各种人员的需要。

计划人员根据范围估算，选择为完成开发工作所需要的技能。并在组织状况(如管理人员、高级软件工程师等)和专业(如通信、数据库、微机等)两方面做出安排。

对于一些规模较小的项目(1 个人年或者更少)，只要向专家做些咨询，也许一个人就可以完成所有的软件工程步骤。对一些规模较大的项目，在整个软件生存期中，各种人员的参与情况是不一样的。图 9.8 画出了各类不同的人员随开发工作的进展在软件工程各个阶段的参与情况的典型曲线。

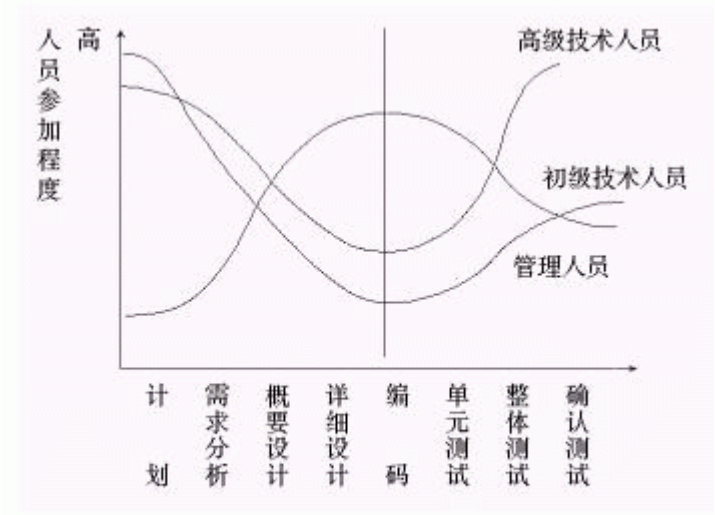


图9.8 管理人员与技术人员的参与情况

一个软件项目所需要的人数只能在对开发的工作量做出估算之后才能决定。

② 硬件资源

硬件是作为软件开发项目的一种工具而投入的，可考虑三种硬件资源：

§ 宿主机(Host machine)——软件开发时使用的计算机及外围设备；
§ 目标机(Target machine)——运行已开发成功软件的计算机及外围设备；
§ 其它硬件设备——专用软件开发时需要的特殊硬件资源；宿主机连同必要的软件工具构成一个软件开发系统。通常这样的开发系统能够支持多种用户的需要，且能保持大量的由软件开发小组成员共享的信息。但在许多情况下，除了那些很大的系统之外，不一定非要配备专门的开发系统。因此，所谓硬件资源，可以认为是对现存计算机系统的使用，宿主机与目标机可以是同一种机型。

可以定义系统中其它的硬件元素为软件开发的资源。例如，在开发自动排版软件的过程中，可能需要一台照相排版机。所有硬件元素都应当由计划人员指定。

③ 软件资源

软件在开发期间使用了许多软件工具来帮助软件的开发。软件工程人员使用在许多方面都类似于硬件工程人员所使用的 CAD / CAE 工具的软件工具集。这种软件工具集叫做计算机辅助软件工程（CASE）。主要的软件工具可做如下分类。

§ 业务系统计划工具——业务系统计划工具借助特定的“元语言”建立一个组织的战略信息需求的模型，导出特定的信息系统。这些工具要解答一些简单但重要的问题，例如，业务关键数据从何处来？这些信息又向何处去？如何使用它们？当它们在业务系统中传递时又如何变换？要增加什么样的新信息？

§ 项目管理工具——项目管理人员使用这些工具可生成关于工作量、成本及软件项目持续时间的估算。定义开发策略及达到这一目标的必要的步骤。计划可行

的项目进程安排。以及持续地跟踪项目的实施。此外，管理人员还可使用工具收集建立软件开发生产率和产品质量的那些度量数据。

§ 支持工具——支持工具可以分类为文档生成工具、网络系统软件、数据库、电子邮件、通报板，以及在开发软件时控制和管理所生成信息的配置管理工具。

§ 分析和设计工具——分析和设计工具可帮助软件技术人员建立目标系统的分析模型和设计模型。这些工具还帮助人们进行模型质量的评价。它们靠对每一个模型进行执行一致性和有效性的检验，帮助软件技术人员在错误扩散到程序中之前排除之。

§ 编程工具——系统软件实用程序、编辑器、编译器及调试程序都是 CASE 中必不可少的部分。而除这些工具之外，还有一些新的编程工具。面向对象的程序设计工具、第四代程序生成语言。高级数据库查询系统，及一大批 PC 工具(如表格软件）。

§ 组装和测试工具——测试工具为软件测试提供了各种不同类型和级别的支持。有些工具，像路径覆盖分析器为测试用例设计提供了直接支持，并在测试的早期使用。其它工具，像自动回归测试和测试数据生成工具，在组装和确认测试时使用，它们能帮助减少在测试过程中所需要的工作量。

§ 原型化和模拟工具——原型化和模拟工具是一个很大的工具集，它包括的范围从简单的窗口画图到实时嵌入系统时序分析与规模分析的模拟产品。原型化工具把注意力集中在建立窗口和为使用户能够了解一个信息系统或工程应用的输入 / 输出域而提出的报告。使用模拟工具可建立嵌入式的实时应用，例如，为一架飞机建立航空控制系统的模型。在系统建立之前，可以对用模拟工具建立起来的模型进行分析，对系统的运行时间性能进行评价。

§ 维护工具——维护工具可以帮助分解一个现存的程序并帮助软件技术人员理解这个程序。软件技术人员必须利用直觉、设计观念和人的智慧来完成逆向工程过程及再工程。

§ 框架工具——这些工具能够提供一个建立集成项目支撑环境（IPSE）的框架。在多数情况，框架工具实际提供了数据库管理和配置管理的能力与一些实用工具，能够把各种工具集成到 IPSE 中。

④ 软件复用性及软件构件库

为了促成软件的复用，以提高软件的生产率和软件产品的质量，可建立可复用的软件部件库。根据需要，对软件部件稍做加工，就可以构成一些大的软件包。这要求这些软件部件应加以编目，以利引用，并进行标准化和确认，以利于应用和集成。 在使用这些软件部件时，有两种情况必须加以注意：

§ 如果有现成的满足要求的软件，应当设法搞到它。因为搞到一个现成的软件所花的费用比重新开发一个同样的软件所花的费用少得多。

§ 如果对一个现存的软件或软件部件，必须修改它才能使用。这时必须多加小心，谨慎对待，因为修改时可能会引出新的问题。而修改一个现存软件所花的费用有时会大于开发一个同样软件所花的费用。

(4) 分解技术

当一个待解决的问题过于复杂时，可以把它进一步分解，直到分解后的子问题变得容易解决为止。然后，分别解决每一个子问题，并将这些子问题的解答综合起来，从而得到原问题的解答。通常，这是我们解决复杂问题的最自然的一种方法。

软件项目估算是一种解决问题的形式，在多数情况下，要解决的问题(对于软件项目来说，就是成本和工作量的估算)非常复杂，想一次性整体解决比较困难。因此，对问题进行分解，把其分解成一组较小的接近于最终解决的可控的子问题，再定义它们的特性。

① LOC 和 FP 估算

LOC 和 FP 是两个不同的估算技术。但两者有许多共同特性。项目计划人员首先给出一个有界的软件范围的叙述，再由此叙述把软件分解成一些小的可分别独立进行估算的子功能。然后对每一个子功能估算其 LOC 或 FP(即估算变量)。接着，把根据以往完成项目得到的(基线)生产率度量(如，LOC / PM 或 FP / PM)用做特定的估算变量，导出子功能的成本或工作量。将子功能的估算进行综合后就能得到整个项目的总估算。

LOC 或 FP 估算技术对于分解所需要的详细程度是不同的。当用 LOC 做为估算变量时，功能分解是绝对必要的且需要达到很详细的程度。而估算功能点所需要的数据是宏观的量，当把 FP 当做估算变量时所需要的分解程度不很详细。

应注意，LOC 是直接估算的，而 FP 是通过估计输入、输出、数据文件、查询和外部接口的数目，以及在表 9.3 中描述的 14 种复杂性校正值间接地确定的。

计划人员可对每一个分解的功能提出一个有代表性的估算值范围。利用历史数据或凭实际经验，对每个功能分别按最佳的、可能的、悲观的三种情况给出 LOC 或 FP 估计值。记作 a、m、b。当这些值的范围被确定之后，也就隐含地指明了估计值的不确定程度。接着计算 LOC 或 FP 的期望值 E。
E ＝ （a＋4m＋b）／6 （加权平均）

确定了估算变量的期望值，就可以用作 LOC 或 FP 的生产率数据。

作为 LOC 和 FP 估算技术的一个实例，考察一个为计算机辅助设计（CAD）应用而开发的软件包。在这个实例中，使用 LOC 做为估算变量。

根据对软件范围的叙述，对软件功能进行分解，识别出主要的几个功能：用户界面和控制功，二维几何分析，三维几何分析，数据库管理，计算机图形显示功能，

外设控制以及设计分析模块。通过下述的分解技术，可得到如表 9.5 所示的估算表。

表 9.5 估算表

功能	a 最佳值	m 可能值	b 悲观值	E 期望值	每行成本 (元 / 行)	生产率 (行 / PM)	成本 (元)	工作量 (PM)
用户接口控制	1800	2400	2650	2340	14	315	32760	7.4
二维几何造型	4100	5200	7400	5380	20	220	107600	24.4
三维几何造型	4600	6900	8600	6800	20	220	136000	30.9
数据库管理	2950	3400	3600	3350	18	240	60300	13.9
终端图形显示	4050	4900	6200	4950	22	200	108900	24.7
外部设备控制	2000	2100	2450	2140	28	140	59920	15.2
设计分析	6600	8500	9800	8400	18	300	151200	28.0
总计				33360			656680	144.5

表中给出了 LOC 的估算范围。计算出的各功能的期望值放入表中的第 4 列。然后对该列垂直求和，得到该 CAD 系统的 LOC 估算值 33360。

从历史数据求出生产率度量和每行成本，即行 / PM 和元 / 行。在表中的成本和工作量这两列的值分别用 LOC 的期望值 E 与元 / 行相乘，及用 LOC 的期望值 E 与行 / PM 相除得到。因此可得，该项目总成本的估算值为 657,000 元，总工作量的估算值为 145 人月(PM）。

② 工作量估算

每一项目任务的解决都需要花费若干人日、人月或人年。每一个工作量单位都对应于一定的货币成本，从而可以由此做出成本估算。

类似于 LOC 或 FP 技术，工作量估算开始于从软件项目范围抽出软件功能。接着给出为实现每一软件功能所必须执行的一系列软件工程任务，包括需求分析、设计、编码和测试。表 9.6 中的表格部分表示各个软件功能和相关的软件工程任务。

计划人员针对每一软件功能，估算完成各个软件工程任务所需要的工作量(如人月)，并记在表 9.6 表格的中心部分。同时，把劳动费用率(即单位工作量成本)加到每个软件工程任务上。最后，计算每一个功能及软件工程任务的工作量和成本。如果工作量估算不依赖 LOC 或 FP 估算，那么，就可得到两组能进行比较和调和的成本与工作量估算。如果这两组估算值合理地一致，则估算值是可靠的。如果估算的结果不一致，就有必要做进一步的检查与分析。

表 9.6 工作量矩阵

	任 务	总计
功 能	工作量估算（人月）	
总计		
费用率（元）		
成本（元）		

以上面所介绍的 CAD 软件为例，列出它的一个完全的工作量估算表(如表 9.7 所示)。表中对每一个软件功能提供了用人月(PM)表示的每一项软件工程任务的工作量估算值。横向和纵向的总计给出所需要的工作量。

表 9.7 工作量估算表

任务 / 功能	需求分析	设 计	编 码	测 试	总 计
用户接口控制	1.0	2.0	0.5	3.5	7.0
二维几何造型	2.0	10.0	4.5	9.5	26.0
三维几何造型	2.5	12.0	6.0	11.0	31.5
数据库管理	2.0	6.0	3.0	4.5	15.0
图形显示功能	1.5	11.0	4.0	10.5	27.5
外设控制功能	1.5	6.0	3.5	5.0	16.0
设计分析功能	4.0	14.0	5.0	7.0	30.0
总计	14.5	61.0	26.5	50.5	152.5
费用率 (元)	5200	4800	4250	4500	
成本 (元)	75400	292800	112625	227250	708075

总工作量估算

总成本估算

除特别指出外，工作量都按人月 (PM)估算。

与每个软件工程任务相关的劳动费用率记入表中费用率(元)这一行，这些数据反映了“负担”的劳动成本，即包括公司开销在内的劳动成本。

如果工作量估算法与 LOC 估算法所得结果之间的一致性很差，就必须重新确定估算所使用的信息。如果要追寻产生差距的原因，不外乎以下两个原因之一：

§ 计划人员没有充分了解或误解了项目的范围；

§ 用于 LOC 估算的生产率数据不适合于本项目，过时了(即使用这些数据不能正确反映软件开发机构的情况)，或者是误用了。

计划人员必须确定产生差距的原因再来协调估算结果。

5、软件开发成本估算

软件开发成本主要是指软件开发过程中所花费的工作量及相应的代价。它不同于其它物理产品的成本，它不包括原材料和能源的消耗，主要是人的劳动的消耗。人的劳动消耗所需代价就是软件产品的开发成本。另一方面，软件产品开发成本的计算方法不同于其它物理产品成本的计算。软件产品不存在重复制造过程，它的开发成本是以一次性开发过程所花费的代价来计算的。因此软件开发成本的估算，应是从软件计划、需求分析、设计、编码、单元测试、组装测试到确认测试，整个软件开发全过程所花费的代价作为依据的。

(1) 软件开发成本估算方法

对于一个大型的软件项目，要进行一系列的估算处理。主要靠分解和类推的手段进行。基本估算方法分为三类。

① 自顶向下的估算方法：这种方法的主要思想是从项目的整体出发,进行类推。即估算人员根据以前已完成项目所消耗的总成本(或总工作量),来推算将要开发的软件的总成本(或总工作量)，然后按比例将它分配到各开发任务单元中去。

这种方法的优点是估算工作量小,速度快。缺点是对项目中的特殊困难估计不足，估算出来的成本盲目性大，有时会遗漏被开发软件的某些部分。

② 自底向上的估计法：这种方法的主要思想是把待开发的软件细分，直到每一个子任务都已经明确所需要的开发工作量，然后把它们加起来，得到软件开发的总工作量。这是一种常见的估算方法。它的优点是估算各个部分的准确性高。缺点是缺少各项子任务之间相互联系所需要的工作量,还缺少许多与软件开发有关的系统级工作量（配置管理、质量管理、项目管理）。所以往往估算值偏低，必须用其它方法进行检验和校正。

③ 差别估计法：这种方法综合了上述两种方法的优点，其主要思想是把待开发的软件项目与过去已完成的软件项目进行类比,从其开发的各个子任务中区分出类似的部分和不同的部分。类似的部分按实际量进行计算，不同的部分则采用相应的方法进行估算。这种的方法的优点是可以提高估算的准确程度，缺点是不容易明确“类似”的界限。

(2) 专家判定技术

专家判定技术就是由多位专家进行成本估算。由于单独一位专家可能会有种种偏见，譬如有乐观的、悲观的、要求在竞争中取胜的、让大家都高兴的种种愿望及政治因素等。因此，最好由多位专家进行估算，取得多个估算值。Rand 公司提出 Deiphi 技术，作为统一专家意见的方法。用 Deiphi 技术可得到极为准确的估算值。

Deiphi 技术的步骤是：

① 组织者发给每位专家一份软件系统的规格说明书（略去名称和单位） 和一张记录估算值的表格，请他们进行估算。

② 专家详细研究软件规格说明书的内容，对该软件提出三个规模的估算值，即：
ai —— 该软件可能的最小规模（最少源代码行数）；

mi —— 该软件最可能的规模（最可能的源代码行数）；

bi —— 该软件可能的最大规模（最多源代码行数）。
无记名地填写表格，并说明做此估算的理由。在填表的过程中，专家互相不进行讨论但可以向组织者提问。

③ 组织者对专家们填在表格中的答复进行整理，做以下事情：

a) 计算各位专家（序号为 i，i＝1,2,…,n，共 n 位专家）的估算期望值 Ei：，并综合各位专家估算值的期望中值 E：

$$E_i = \frac{a_i + 4m_i + b_i}{6} \qquad E = \frac{1}{n}\sum_{i=1}^{n} E_i$$

b) 对专家的估算结果进行分类摘要。

④ 在综合专家估算结果的基础上，组织专家再次无记名地填写表格。 然后比较两次估算的结果。若差异很大，则要通过查询找出差异的原因。

⑤ 上述过程可重复多次。最终可获得一个得到多数专家共识的软件规模（源代码行数）。在此过程中不得进行小组讨论。

最后，通过与历史资料进行类比，根据过去完成软件项目的规模和成本等信息，推算出该软件每行源代码所需要的成本。然后再乘以该软件源代码行数的估算值，就可得到该软件的成本估算值。

此方法的缺点是人们无法利用其他参加者的估算值来调整自己的估算值。宽带 Deiphi 技术克服了这个缺点。在专家正式将估算值填入表格之前，由组织者召集小组会议，专家们与组织者一起对估算问题进行讨论，然后专家们再无记名填表。组织者对各位专家在表中填写的估算值进行综合和分类后，再召集会议，请专家们对其估算值有很大变动之处进行讨论，请专家们重新无记名填表。这样适当重复几次，得到比较准确的估计值。

由于增加了协商的机会，集思广益，使得估算值更趋于合理。

(3) 软件开发成本估算的早期经验模型

软件开发成本估算是依据开发成本估算模型进行估算的。开发成本估算模型通常采用经验公式来预测软件项目计划所需要的成本、工作量和进度数据。还没有一

种估算模型能够适用于所有的软件类型和开发环境，从这些模型中得到的结果必须慎重使用。

① IBM 模型

1977 年，IBM 的 Walston 和 Felix 提出了如下的估算公式：

$E = 5.2 \times L^{0.91}$，…L 是源代码行数(以 KLOC 计)，E 是工作量(以 PM 计)

$D = 4.1 \times L^{0.36} = 14.47 \times E^{0.35}$，…D 是项目持续时间(以月计)

$S = 0.54 \times E^{0.6}$，…S 是人员需要量(以人计)

$DOC = 49 \times L^{1.01}$。…DOC 是文档数量(以页计)

在此模型中，一般指一条机器指令为一行源代码。一个软件的源代码行数不包括程序注释、作业命令、调试程序在内。对于非机器指令编写的源程序，例如汇编语言或高级语言程序，应转换成机器指令源代码行数来考虑。 IBM 模型是一个静态单变量模型，但不是一个通用的公式。 在应用中有时要根据具体实际情况，对公式中的参数进行修改。这种修改必须拥有足够的历史数据，在明确局部的环境之后才能做出。

② Putnam 模型

这是 1978 年 Putnam 提出的模型，是一种动态多变量模型。它是假定在软件开发的整个生存期中工作量有特定的分布。这种模型是依据在一些大型项目(总工作量达到或超过 30 个人年)中收集到的工作量分布情况而推导出来的，但也可以应用在一些较小的软件项目中。

Putnam 模型可以导出一个“软件方程”，把已交付的源代码(源语句)行数与工作量和开发时间联系起来。其中，td 是开发持续时间(以年计)，K 是软件开发与维护在内的整个生存期所花费的工作量(以人年计)，L 是源代码行数(以 LOC 计)，Ck 是技术状态常数，它反映出“妨碍程序员进展的限制”，并因开发环境而异。其典型值的选取如表 9.8 所示。

$$L = Ck \cdot K^{\frac{1}{3}} \cdot td^{\frac{4}{3}}$$

表 9.8 技术状态常数 Ck 的取值

Ck 的典型值	开发环境	开发环境举例
2000	差	没有系统的开发方法，缺乏文档和复审，批处理方式。
8000	好	有合适的系统开发方法，有充分的文档和复审，交互执行方式。
11000	优	有自动开发工具和技术。

(4) COCOMO 模型（COnstructive COst MOdel）

这是由 TRW 公司开发。Boehm 提出的结构型成本估算模型。是一种精确、易于使用的成本估算方法。在该模型中使用的基本量有以下几个：DSI（源指令条数）定义为代码或卡片形式的源程序行数。若一行有两个语句，则算做一条指令。它包括作业控制语句和格式语句，但不包括注释语句。KDSI＝1000DSI。MM（度量单位为人月）表示开发工作量。TDEV（度量单位为月）表示开发进度。它由工作量决定。

① 软件开发项目的分类

在 COCOMO 模型中，考虑开发环境，软件开发项目的总体类型可分为三种：组织型(Organic)、嵌入型(Embadded)和介于上述两种软件之间的半独立型(Semidetached)。

② COCOMO 模型的分类

COCOMO 模型按其详细程度分成三级：即基本 COCOMO 模型、中间 COCOMO 模型、详细 COCOMO 模型。基本 COCOMO 模型是一个静态单变量模型，它用一个以已估算出来的源代码行数(LOC)为自变量的(经验)函数来计算软件开发工作量。中间 COCOMO 模型则在用 LOC 为自变量的函数计算软件开发工作量(此时称为名义工作量)的基础上，再用涉及产品、硬件、人员、项目等方面属性的影响因素来调整工作量的估算。详细 COCOMO 模型包括中间 COCOMO 模型的所有特性，但用上述各种影响因素调整工作量估算时，还要考虑对软件工程过程中每一步骤(分析、设计等)的影响。

③ 基本 COCOMO 模型

基本 COCOMO 模型的工作量和进度公式如表 9.9 所示。

表 9.9 基本 COCOMO 模型的工作量和进度公式

总体类型	工作量	进度
组织型	$MM=2.4(KDSI)^{1.05}$	$TDEV=2.5(MM)^{0.38}$
半独立型	$MM=3.0(KDSI)^{1.12}$	$TDEV=2.5(MM)^{0.35}$
嵌入型	$MM=3.6(KDSI)^{1.20}$	$TDEV=2.5(MM)^{0.32}$

利用上面公式，可求得软件项目，或分阶段求得各软件任务的开发工作量和开发进度。

④ 中间 COCOMO 模型

进一步考虑以下 15 种影响软件工作量的因素，通过定下乘法因子，修正 COCOMO 工作量公式和进度公式，可以更合理地估算软件(各阶段)的工作量和进度。

中间 COCOMO 模型的名义工作量与进度公式如表 9.10 所示。

表 9.10 中间 COCOMO 模型的名义工作量与进度公式

总体类型	工作量	进度
组织型	$MM=3.2(KDSI)^{1.05}$	$TDEV=2.5(MM)^{0.38}$
半独立型	$MM=3.0(KDSI)^{1.12}$	$TDEV=2.5(MM)^{0.35}$
嵌入型	$MM=2.8(KDSI)^{1.20}$	$TDEV=2.5(MM)^{0.32}$

对 15 种影响软件工作量的因素 fi 按等级打分，如表 9.11 所列。此时，工作量计算公式改成：

⑤ 详细 COCOMO 模型

详细COCOMO模型的名义工作量公式和进度公式与中间COCOMO模型相同。但分层、分阶段给出工作量因素分级表(类似于上表）。针对每一个影响因素，按模块层、子系统层、系统层，有三张不同的工作量因素分级表，供不同层次的估算使用。每一张表中工作量因素又按开发各个不同阶段给出。

例如，关于软件可靠性(RELY)要求的工作量因素分级表(子系统层)，如表 9.12 所示。使用这些表格，可以比中间 COCOMO 模型更方便、更准确地估算软件开发工作量。

表 9.12 软件可靠性工作量因素分级表（子系统层）

阶段 / RELY 级别	需求和产品设计	详细设计	编码及单元测试	集成及测试	综合
非常低	0.80	0.80	0.80	0.60	0.75
低	0.90	0.90	0.90	0.80	0.88
正常	1.00	1.00	1.00	1.00	1.00
高	1.10	1.10	1.10	1.30	1.15
非常高	1.30	1.30	1.30	1.70	1.40

6、成本－效益分析

成本－效益分析的目的，是从经济角度评价开发一个新的软件项目是否可行。

成本－效益分析首先是估算新软件系统的开发成本，然后与可能取得的效益(有形的和无形的)进行比较权衡。有形的效益可以用货币的时间价值、投资回收期、纯收入、投资回收率等指标进行度量。无形的效益主要是从性质上、心理上进行衡量，很难直接进行量上的比较。无形的效益在某些情形下会转化成有形的效益。例如，一个高质量的设计先进的软件可以使用户更满意，从而影响到其他潜在的

用户也会喜欢它，一旦需要时就会选择购买它，这样使得无形的效益转化成有形的效益。

系统的经济效益等于因使用新系统而增加的收入加上使用新系统可以节省的运行费用。运行费用包括操作员人数、工作时间、消耗的物资等。

(1) 几种度量效益的方法

① 货币的时间价值

成本估算的目的是要求对项目投资。但投资在前，取得效益在后。因此要考虑货币的时间价值。通常用利率表示货币的时间价值。设年利率为 i，现已存入 P 元，则 n 年后可得钱数为：F = (1 + i) n，这就是 P 元钱在 n 年后的价值。 反之，若 n 年后能收入 F 元，那么这些钱现在的价值是：

$$P = \frac{F}{(1+i)^n}$$

例如，在工程设计中用 CAD 系统来取代大部分人工设计工作，每年可节省 9.6 万元。若软件生存期为 5 年，则 5 年可节省 48 万元。开发这个 CAD 系统共投资了 20 万元。我们不能简单地把 20 万元与 48 万元相比较。因为前者是现在投资的钱，而后者是 5 年以后节省的钱。需要把 5 年内每年预计节省的钱折合成现在的价值才能进行比较。

设年利率是 5%，利用上面计算货币现在价值的公式，可以算出引入 CAD 系统后，每年预计节省的钱的现在价值，参看表 9.13

表 9.13 货币的时间价值

年份	将来值 (万)	(1+i)^n	现在值 (万)	累计的现在值 (万)
1	9.6	1.05	9.1429	9.1429
2	9.6	1.1025	8.7075	17.8513
3	9.6	1.1576	8.2928	26.1432
4	9.6	1.2155	7.8979	34.0411
5	9.6	1.2763	7.5219	41.5630

② 投资回收期

投资回收期是衡量一个开发工程价值的经济指标。所谓投资回收期就是使累计的经济效益等于最初的投资所需的时间。投资回收期越短，就能越快获得利润。因此这项工程就越值得投资。例如，引入 CAD 系统两年后可以节省 17.85 万元，比最初的投资还少 2.15 万元，但第三年可以节省 8.29 万元，则 2.15 / 8.29 = 0.259。因此，投资回收期是 2.259 年。

③ 纯收入

工程的纯收入是衡量工程价值的另一项经济指标。所谓纯收入就是在整个生存期之内系统的累计经济效益(折合成现在值)与投资之差。例如，引入 CAD 系统之后，5 年内工程的纯收入预计是 41.563－20＝21.563（万元）。这相当于比较投资一个待开发的软件项目后预期可取得的效益和把钱存在银行里(或贷款给其它企业)所取得的收益，到底孰优孰劣。如果纯收入为零，则工程的预期效益与在银行存款一样。但开发一个软件项目有风险，从经济观点看，这项工程可能是不值得投资的。如果纯收入小于零，那么显然这项工程不值得投资。只有当纯收入大于零，才能考虑投资。

④ 投资回收率

把钱存在银行里，可以用年利率来衡量利息的多少。类似地，用投资回收率来衡量投资效益的大小。已知现在的投资额 P，并且已经估算出将来每年可以获得的经济效益 Fk，以及软件的使用寿命 n，k=1,2,...,n。则投资回收率 j，可用如下的方程来计算：

$$P = \frac{F_1}{(1+j)^1} + \frac{F_2}{(1+j)^2} + \frac{F_3}{(1+j)^3} + \Lambda\,\Lambda + \frac{F_n}{(1+j)^n}$$

这相当于把数额等于投资额的资金存入银行，每年年底从银行取回的钱等于系统每年预期可以获得的效益。在时间等于系统寿命时，正好把在银行中的钱全部取光。此时的年利率是多少呢？就等于投资回收率。

(2) 成本－效益的分析

系统的效益分析随系统的特性而异。大多数数据处理系统的基本目标是开发具有较大信息容量、更高的质量、更及时、组织得更好的系统。因此，效益集中在信息存取和它对用户环境的影响上面。与工程－科学计算软件及基于微处理器的产品相关的效益在本质上可能不大相同。 新系统的效益与系统的工作方式有关。仍以前面所说的代替人工设计过程的计算机辅助设计(CAD)系统为例。分析员对现行系统(人工设计)和待开发系统(CAD)定义可度量的特性，选定产生最终详细图纸的时间 t-draw 作为一个可度量的特性。且分析员发现，CAD 系统产生的缩减比为 1 / 4。为进一步量化这种效益，应确定以下数据：t-draw：平均绘图时间 ＝ 4 小时；c：每绘图一小时的成本 ＝ 20.00 元；n：每年绘图的数量 ＝ 8000；p：CAD 系统中已完成绘图的百分比 ＝ 60%；利用以上已知数据，每年节省费用的估算值，即所得到的效益为： 节省的绘图费用 ＝ 缩减比 ＊t-draw ＊ n ＊ c ＊ p ＝ 96000 元 / 年。

其它由 CAD 系统而得到的有形效益将以类似的方式进行处理。而无形的效益(如较好的设计质量、较高的雇员素质)可以被赋予货币价值。

软件系统开发的成本如表 9.14 所列。

表 9.14 信息系统可能的费用

筹办费用	咨询费	实际设备购置或租用设备费
	设备安装费	设备场所改建费(空调、安全设施等)
	资本	与筹办相关的管理和人员的费用
开办费用	操作系统软件的费用	通信设备安装费用(电话线、数据线等)
	开办人员的费用	人员寻找与聘用活动所需的费用
	破坏其它机构所需的费用	指导开办活动所需的管理费用
与项目有关的费用	应用软件购置费	为适应局域系统修改软件的费用
	公司内应用系统开发所需的人员工资、经常性开销等	数据收集和建立数据收集过程所需的费用
	准备文档所需的费用	培训用户人员使用应用系统的费用
	开发管理费	
运行费用	系统维护费用(硬件、软件和设备)	租借费用(电费、电话费等)
	硬件折旧费	信息系统管理、操作及计划活动中涉及人员的费用

分析员可以估算每一项的成本，然后用开发费用和运行费用来确定投资的偿还、损益两平点和投资回收期。图 9.9 说明了前面所介绍的 CAD 系统实例的特性。假设每年可节约总费用的估计值为 96000 元，总开发（或购买）费用为 204000 元，年度费用估计为 32000 元。则从图 9.0 可知，投资回收期大约需要 3.1 年。实际上，投资的偿还可以用更详细的分析方法来确定，即货币的时间价值、税收的影响及其它潜在的对投资的使用。再把无形的效益考虑在内，上级管理部门就可以决策，在经济上是否值得开发这个系统。

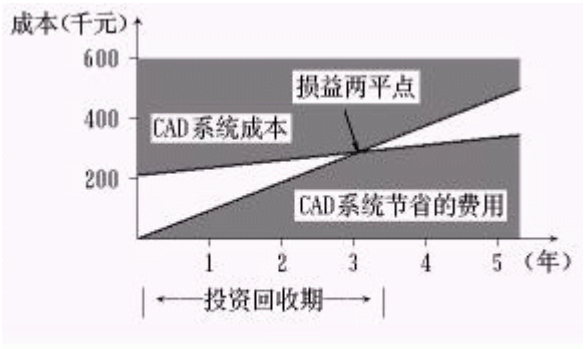


图 9.9 成本—效益分析

7、风险分析

风险分析实际上是 4 个不同的活动： 风险识别，风险估计，风险评价和风险驾驭。

(1) 风险识别

风险识别就是要系统地确定对项目计划(估算、进度、资源分配)的威胁，通过识别已知的或可预测的风险，就可能设法避开风险或驾驭风险。

① 风险类型

用各种不同的方法对风险进行分类是可能的。从宏观上来看，可将风险分为项目风险、技术风险和商业风险。

项目风险威胁到项目计划，一旦项目风险成为现实，可能会拖延项目进度，增加项目的成本。项目风险是指潜在的预算、进度、个人(包括人员和组织)、资源、用户和需求方面的问题，以及它们对软件项目的影响。项目复杂性、规模和结构的不确定性也构成项目的(估算)风险因素。

技术风险威胁到待开发软件的质量和预定的交付时间。如果技术风险成为现实，开发工作可能会变得很困难或根本不可能。技术风险是指潜在的设计、实现、接口、检验和维护方面的问题。此外，规格说明的多义性、技术上的不确定性、技术陈旧、最新技术(不成熟)也是风险因素。技术风险之所以出现是由于问题的解决比我们预想的要复杂。

商业风险威胁到待开发软件的生存能力。5 种主要的商业风险是：

§ 建立的软件虽然很优秀但不是市场真正所想要的（市场风险）；

§ 建立的软件不再符合公司的整个软件产品战略（策略风险）；

§ 建立了销售部门不清楚如何推销的软件；

§ 由于重点转移或人员变动而失去上级管理部门的支持（管理风险）；

§ 没有得到预算或人员的保证（预算风险）。

特别要注意，有时对某些风险不能简单地归类，而且某些风险事先是无法预测的。

② 风险项目检查表

识别风险的一种最好的方法就是利用一组提问来帮助项目计划人员了解在项目和技术方面有哪些风险。因此，Boehm 建议使用一个“风险项目检查表”，列出所有可能的与每一个风险因素有关的提问。从如下几个方面识别已知的或可预测的风险。

§ 产品规模——与待开发或要修改的软件的产品规模(估算偏差、产品用户、需求变更、复用软件、数据库)相关的风险。

§ 商业影响——与管理和市场所加之的约束(公司收益、上级重视、符合需求、用户水平、产品文档、政府约束、成本损耗、交付期限)有关的风险。

§ 客户特性——与客户的素质(技术素养、合作态度、需求理解)以及开发者与客户定期通信(技术评审、通信渠道)能力有关的风险。

§ 过程定义——与软件过程定义与组织程度以及开发组织遵循的程度相关的风险。

§ 开发环境——与用来建造产品的工具(项目管理、过程管理、分析与设计、编译器及代码生成器、测试与调试、配置管理、工具集成、工具培训、联机帮助与文档)的可用性和质量相关的风险。

§ 建造技术——与待开发软件的复杂性及系统所包含技术的“新颖性”相关的风险。

§ 人员数量及经验——与参与工作的软件技术人员的总体技术水平(优秀程度、专业配套、数量足够、时间窗口)及项目经验(业务培训、工作基础)相关的风险。

③ 全面评估项目风险

下面的问题是通过调查世界各地有经验的软件项目管理者而得到的风险数据导出的。这些问题根据它们对于项目成功的相对重要性顺序排列。

§ 高层的软件和客户的管理者是否正式地同意支持该项目?

§ 终端用户是否热心地支持该项目和将要建立的系统/产品?

§ 软件工程组和他们的客户对于需求是否有充分的理解?

§ 客户是否充分地参与了需求定义过程。

§ 终端用户的期望是否实际?

§ 项目的范围是否稳定?

§ 软件工程组是否拥有为完成项目所必须的各种技术人才。

§ 项目的需求是否稳定?

§ 项目组是否具有实现目标软件系统的技术基础和工作经验?

§ 项目组中的人员数量对于执行项目的任务是否足够?

§ 所有客户是否都一致赞同该项目的重要性并支持将要建立的系统/产品 §

只要对这些问题的回答有一个是否定的,就应当制定缓解、监控、驾驭的步骤以避免项目失败。项目处于风险的程度直接与对这些问题否定回答的数目成比例。

④ 风险构成和驱动因素

美国空军在一本关于如何识别和消除风险的手册中给出如何标识影响风险构成的风险驱动因素。风险构成有如下几种：

§ 性能风险——产品是否能够满足需求且符合其使用目的的不确定的程度。

§ 成本风险——项目预算是否能够维持的不确定的程度。

§ 支持风险——软件产品是否易于排错、适应及增强的不确定的程度。

§ 进度风险——项目进度是否能够维持及产品是否能够按时交付的不确定的程度。

如表 9.15 所示，每个风险驱动因素对风险构成的影响可以划分为 4 类：可忽略的、边缘的、危急的、灾难的。

(2) 风险估计

风险估计从两个方面估价每一种风险。一是估计一个风险发生的可能性。一是估价与风险相关的问题出现后将会产生的结果。通常，项目计划人员与管理人员、技术人员一起，进行 4 种风险估计活动：建立一个尺度来表明风险发生的可能性；描述风险的后果；估计风险对项目和产品的影响；指明风险估计的正确性以便消除误解。

① 建立风险表

风险表的示例如表 9.16 所示。第 1 列列出风险(不论多么细微)，可以利用风险项目检查表的条目来给出。每一个风险在第 2 列加以分类，在第 3 列给出风险发生概率，第 4 列是利用表 9.15 给出的对风险产生影响的评价。这要求对 4 种风险构成(性能、支持、成本、进度)的影响类别求平均值，得到一个整体的影响值。

表 9.16 风险表

风 险	类 别	概率	影响	RMMM
规模估算可能非常低	PS 产品规模风险	60%	2	
用户数量大大超过计划	PS …々	30%	3	
复用程度低于计划	PS …々	70%	2	风险
最终用户抵制该系统	BU 商业风险	40%	3	缓解
交付期限将被紧缩	BU …々	50%	2	监控
资金将会流失	CU 客户特性风险	40%	1	驾驭
用户将改变需求	PS 产品规模风险	80%	2	计划
技术达不到预期的效果	TE 建造技术风险	30%	1	
缺少对工具的培训	DE 开发环境风险	80%	3	
参与人员缺乏经验	ST 人员规模与经验风险	30%	2	
参与人员流动比较频繁	ST …々	60%	2	
……				

风险出现概率可以使用从过去项目、直觉或其它信息收集来的度量数据进行统计分析估算出来。例如，由 45 个项目中收集的度量表明，有 37 个项目遇到的用户要求变更次数达到 2 次。 作为预测，新项目将遇到的极端的要求变更次数的概率是 37 / 45＝0.82，因而，这是一个极可能的风险。

一旦完成了风险表前四列的内容，就可以根据概率和影响进行排序。高发生概率和高影响的风险移向表的前端，低概率低影响的风险向后移动，完成第一次风险优先排队。

项目管理人员研究已排序的表，定义一条截止线(Cutoff line)，这条截止线(在表中某一位置的一条横线)表明，位于线上部分的风险将给予进一步关注而位于线下部分的风险需要再评估以完成第二次优先排队。

参看图 9.10，风险影响和发生概率对驾驭参与有不同的影响。一个具有较高的影响但发生概率极低的风险应当不占用很多有效管理时间。然而，具有中等到高概率的高影响的风险和具有高概率的低影响的风险，就必须进行风险的分析。

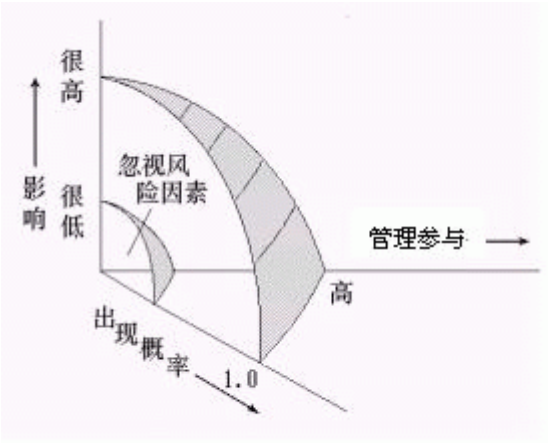


图 9.10 风险与管理参与

② 风险评价

在进行风险评价时，可建立一系列三元组：[ri, li, xi]，其中，ri 是风险，li 是风险出现的可能性(概率)，而 xi 是风险产生的影响。在做风险评价时，应进一步审查在风险估计时所得到的估计的准确性，尝试对已发现的风险进行优先排队，并着手考虑控制和／或消除可能出现风险的方法。

在做风险评价时常采用的一个非常有效的方法就是定义风险参照水准。对于大多数软件项目来说，性能、支持、成本、进度就是典型的风险参照水准。就是说，对于成本超支、进度延期、性能降低、支持困难，或它们的某种组合，都有一个水准值，超出它就会导致项目被迫终止。如果风险的某种组合所产生的问题导致一个或多个这样的参照水准被超出，工作就要中止。在软件风险分析的上下文中，一个风险参照水准就有一个点，叫做参照点或崩溃点。在这个点上，要公平地给出可接受的判断，看是继续执行项目工作，还是终止它们（出的问题太大）。

图 9.11 用图示表示这种情况。如果因为风险的某一组合造成问题，导致项目成本超支和进度延迟，一系列的参照点构成一条曲线，超出它时就会引起项目终止(黑色阴影区域)。在参照点上，要对做出是继续进行还是终止的判断公正地加权。

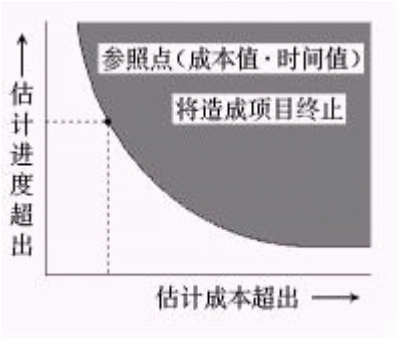


图 9.11 风险参照水准曲线

实际上，参照点能在图上被表示成一条平滑的曲线的情况很少。在多数情况中，它是一个区域，在此区域中存在许多不确定性的范围，在这些范围内想做出基于参照值组合的管理判断往往是不可能的。

因此，在做风险评价时，我们按以下步骤执行：

§ 定义项目的各种风险参照水准；

§ 找出在各 [ri, li, xi] 和各参照水准之间的关系；

§ 预测一组参照点以定义一个项目终止区域，用一条曲线或一些不确定性区来界定；

§ 预测各种复合的风险组合将如何影响参照水准。

(3) 风险驾驭和监控

为了执行风险驾驭与监控活动，必须考虑与每一风险相关的三元组(风险描述、风险发生概率、风险影响)，它们构成风险驾驭(风险消除)步骤的基础。例如，假如人员的频繁流动是一项风险 ri，基于过去的历史和管理经验，频繁流动可能性的估算值 li 为 0.70，而影响 xi 的估计值是：项目开发时间增加 15%，总成本增加 12%。为了缓解这一风险，项目管理必须建立一个策略来降低人员的流动造成的影响。可采取的风险驾驭步骤如下：

§ 与现有人员一起探讨人员流动的原因(如，工作条件差，收入低，人才市场竞争等)；

§ 在项目开始前，把缓解这些原因的工作列入管理计划中。

§ 当项目启动时，做好人员流动会出现的准备。采取一些技术以确保人员一旦离开后项目仍能继续；

§ 建立良好的项目组织和通信渠道，以使大家都了解每一个有关开发活动的信息；

§ 制定文档标准并建立相应机制，以保证文档能够及时建立；

§ 对所有工作组织细致的评审，使大多数人能够按计划进度完成自己的工作；

§ 对每一个关键性的技术人员，要培养后备人员。

这些风险驾驭步骤带来了额外的项目成本。例如，培养关键技术人员的后备需要花钱、花时间。因此，当通过某个风险驾驭步骤而得到的收益被实现它们的成本超出时，要对风险驾驭部分进行评价，进行传统的成本－效益分析。

对于一个大型的软件项目，可能识别 30－40 项风险。如果每一项风险有 3－7 个风险驾驭步骤，那么风险驾驭本身也可能成为一个项目。 正因为如此，我们把 Pareto 的 80 / 20 规则用到软件风险上来。经验表明,所有项目风险的 80%(即可能导致项目失败的 80% 的潜在因素)能够通过 20% 的已识别风险来说明。在早期风险分析步骤中所做的工作可以帮助计划人员确定，哪些风险属于这 20% 之内。由于这个原因，某些被识别过、估计过及评价过的风险可以不写进风险驾驭计划中，因为它们不属于关键的 20%（具有最高项目优先级的风险）。

风险驾驭步骤要写进风险缓解、监控和驾驭计划 RMMM（Risk Mitigation Monitoring and Management Plan）。RMMM 记叙了风险分析的全部工作，并且做为整个项目计划的一部分为项目管理人员所使用。

一旦制定出 RMMM 且项目已开始执行，风险缓解与监控就开始了。风险缓解是一种问题回避活动，风险监控是一种项目追踪活动，它有三个主要目标：

§ 判断一个预测的风险事实上是否发生了；

§ 确保针对某个风险而制定的风险消除步骤正在合理地使用；

§ 收集可用于将来的风险分析的信息。

多数情况下，项目中发生的问题总能追踪到许多风险。风险监控的另一项工作就是要把“责任”（什么风险导致问题发生）分配到项目中去。

风险分析需要占用许多有效的项目计划工作量。识别，估计，评价，管理和监控都需要时间，但这些工作量花得值得。孙子曾经说过：“知己知彼，百战不殆。”

8、进度安排

软件开发项目的进度安排有两种考虑方式：

（1）系统最终交付日期已经确定，软件开发部门必须在规定期限内完成；

（2）系统最终交付日期只确定了大致的年限，最後交付日期由软件开发部门确定。

後一种安排能够对软件项目进行细致分析，最好地利用资源，合理地分配工作，而最后的交付日期可以在对软件进行仔细地分析之后再确定下来；但前一种安排在实际工作中常遇到，如不能按时完成，用户会不满意，甚至还会要求赔偿经济损失，所以必须在规定的期限内合理地分配人力和安排进度。

进度安排的准确程度可能比成本估算的准确程度更重要。软件产品可以靠重新定价或者靠大量的销售来弥补成本的增加，但是进度安排的落空，会导致市场机会的丧失，使用户不满意，而且也会导致成本的增加。

(1) 软件开发小组人数与软件生产率

对于一个小型软件开发项目，一个人就可以完成需求分析、设计、编码和测试工作。随着软件开发项目规模的增大，就需要更多的人共同参与同一软件项目的工作，因此要求由多人组成软件开发组。但是，软件产品是逻辑产品而不是物理产品，当几个人共同承担软件开发项目中的某一任务时，人与人之间必须通过交流来解决各自承担任务之间的接口问题，即所谓通信问题。通信需花费时间和代价，会引起软件错误增加，降低软件生产率。

若两个人之间需要通信，则在这两人之间存在一条通信路径。如果一个软件开发组有 n 个人，每两人之间都需要通信，则总的通信路径有 n*(n-1)/2（条）。假设一个人单独开发软件，生产率是 5000 行 / 人年。若 4 个人组成一个小组共同开发这个软件，则需要 6 条通信路径。若在每条通信路径上耗费的工作量是 250 行 / 人年。则组中每人的生产率降低为：

5000－6×250 / 4 ＝ 5000－375 ＝ 4625 行 / 人年。

从上述简单分析可知，一个软件任务由一个人单独开发，生产率最高；而对于一个稍大型的软件项目，一个人单独开发，时间太长。因此软件开发组是必要的。

有人提出，软件开发组的规模不能太大，人数不能太多，一般在 2～8 人左右为宜。

(2) 任务的确定与并行性

当参加同一软件工程项目的人数不止一人的时候，开发工作就会出现并行情形。图 9.12 显示了一个典型的由多人参加的软件工程项目的任务图。

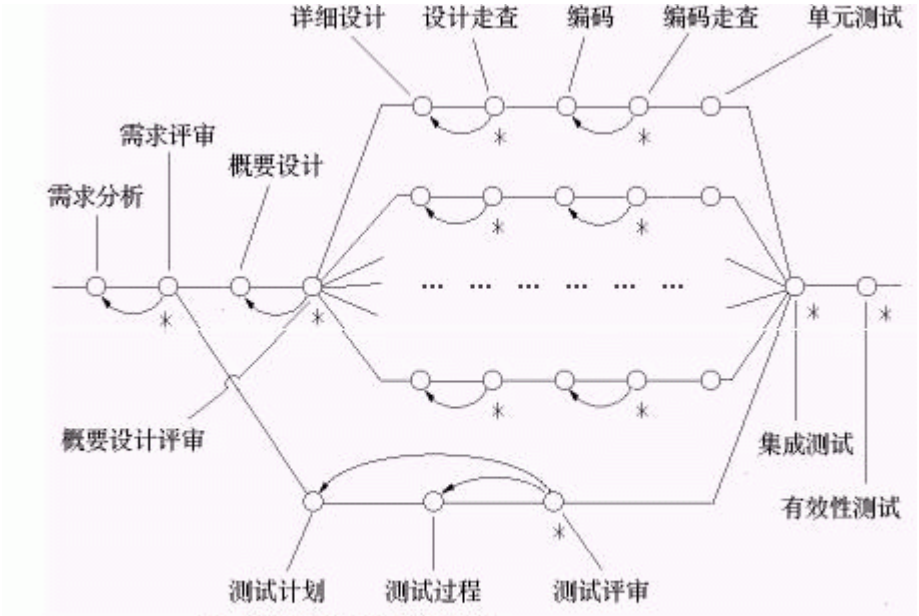


图 9.12 软件项目的并行性

在软件开发过程的各种活动中，第一项任务是进行项目的需求分析和评审，此项工作为以后的并行工作打下了基础。一旦软件的需求得到确认，并且通过了评审，概要设计(系统结构设计和数据设计)工作和测试计划制定工作就可以并行进行。如果系统模块结构已经建立，对各个模块的详细设计、编码、单元测试等工作又可以并行进行。待到每一个模块都已经调试完成，就可以对它们进行组装，并进行组装测试，最后进行确认测试，为软件交付进行确认工作。在图中可以看到，软件开发进程中设置了许多里程碑。里程碑为管理人员提供了指示项目进度的可靠依据。当一个软件工程任务成功地通过了评审并产生了文档之后，一个里程碑就完成了。 软件工程项目的并行性提出了一系列的进度要求。因为并行任务是同时发生的，所以进度计划必须决定任务之间的从属关系，确定各个任务的先后次序和衔接，确定各个任务完成的持续时间。此外，应注意构成关键路径的任务，即若要保证整个项目能按进度要求完成，就必须保证这些任务要按进度要求完成。这样就可以确定在进度安排中应保证的重点。

(3) 制定开发进度计划

图 9.13 给出了在整个定义与开发阶段工作量分配的一种建议方案。这个分配方

案称为 40－20－40 规则。它指出在整个软件开发过程中，编码的工作量仅占 20%，编码前的工作量占 40%，编码后的工作量占 40%。

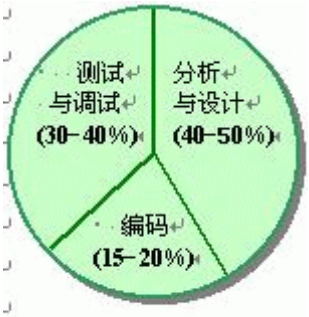


图 9.13 工作量的分配

40－20－40 规则只用来做为一个指南。实际的工作量分配比例必须按照每个项目的特点来决定。一般在计划阶段的工作量很少超过总工作量的 2%～3%，除非是具有高风险的巨额投资的项目。需求分析可能占总工作量的 10%～25%。花费在分析或原型化上面的工作量应当随项目规模和复杂性成比例地增加。通常用于软件设计的工作量在 20%～25%之间。而用在设计评审与反复修改的时间也必须考虑在内。

由于软件设计已经投入了工作量，因此其后的编码工作相对来说困难要小一些，用总工作量的 15%～20%就可以完成。测试和随后的调试工作约占总工作量的 30%～40%。所需要的测试量往往取决于软件的重要程度。

进一步地，由 COCOMO 模型可知，开发进度 TDEV 与工作量 MM 的关系：

$$TDEV = a(MM)^b$$

如果想要缩短开发时间,或想要保证开发进度,必须考虑影响工作量的那些因素。按可减小工作量的因素取值。比较精确的进度安排可利用中间 COCOMO 模型或详细 COCOMO 模型。

(4) 进度安排的方法

软件项目的进度安排与任何一个多任务工作的进度安排基本差不多，因此，只要稍加修改，就可以把用于一般开发项目的进度安排的技术和工具应用于软件项目。

软件项目的进度计划和工作的实际进展情况，需要采用图示的方法描述，特别是表现各项任务之间进度的相互依赖关系。以下介绍几种有效的图示方法。在这几种图示方法中，有几个信息必须明确标明：

- 各个任务的计划开始时间，完成时间；

- 各个任务完成的标志（即○文档编写和△评审）；
- 各个任务与参与工作的人数，各个任务与工作量之间的衔接情况；
- 完成各个任务所需的物理资源和数据资源。

① 甘特图

甘特图用水平线段表示任务的工作阶段;线段的起点和终点分别对应着任务的开工时间和完成时间；线段的长度表示完成任务所需的时间。图 9.14 给出了一个具有 5 个任务的甘特图。如果这 5 条线段分别代表完成任务的计划时间，则在横坐标方向附加一条可向右移动的纵线。它可随着项目的进展，指明已完成的任务(纵线扫过的)和有待完成的任务(纵线尚未扫过的)。我们从甘特图上可以很清楚地看出各子任务在时间上的对比关系。

在甘特图中，每一任务完成的标准，不是以能否继续下一阶段任务为标准，而是必须交付应交付的文档与通过评审为标准。因此在甘特图中，文档编制与评审是软件开发进度的里程碑。甘特图的优点是标明了各任务的计划进度和当前进度，能动态地反映软件开发进展情况。缺点是难以反映多个任务之间存在的复杂的逻辑关系。

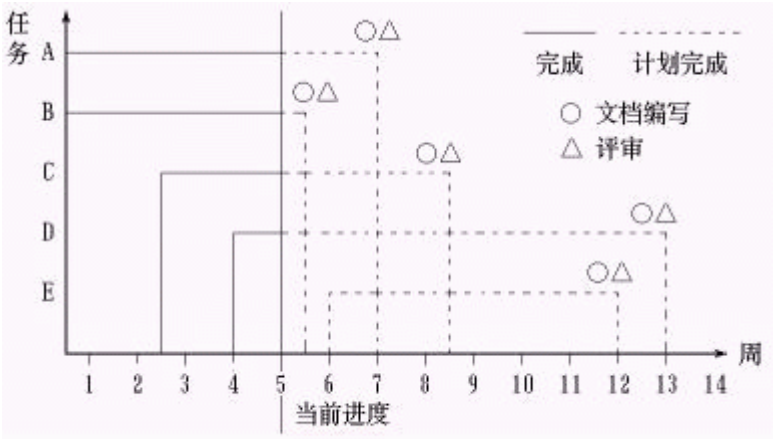


图 9.14 甘特图

② PERT 技术和 CPM 方法

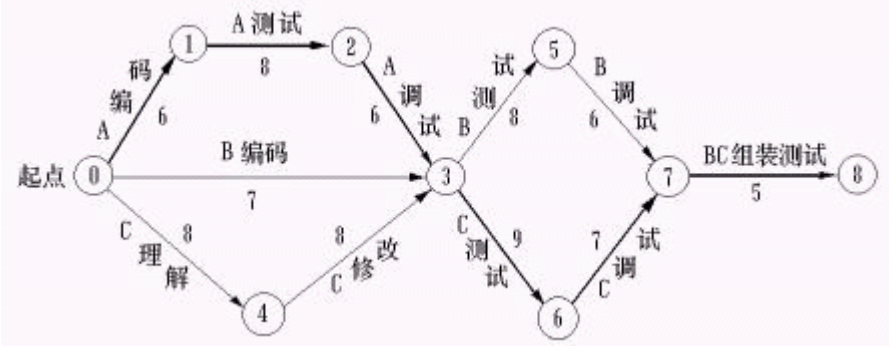
PERT 技术叫做计划评审技术,CPM 方法叫做关键路径法,它们都是安排开发进度,制定软件开发计划的最常用的方法。它们都采用网络图来描述一个项目的任务网络，也就是从一个项目的开始到结束，把应当完成的任务用图或表的形式表示出来。通常用两张表来定义网络图。一张表给出与一特定软件项目有关的所有任务(也称为任务分解结构)，另一张表给出应当按照什么样的次序来完成这些任务(也称为限制表)。

PERT 技术和 CPM 方法都为项目计划人员提供了一些定量的工具，以：

§ 确定关键路径，即决定项目开发时间的任务链。

§ 应用统计模型，对每一个单独的任务确定最可能的开发持续时间的估算值。

§ 计算边界时间，以便为具体的任务定义时间窗口。边界时间的计算对于软件项目的计划调度是非常有用的。



例如，某一开发项目在进入编码阶段之后，考虑安排三个模块 A、B、C 的开发工作。其中，模块 A 是公用模块，模块 B 与 C 的测试有赖于模块 A 调试的完成。模块 C 是利用现成已有的模块，但对它要在理解之后做部分修改。最后直到 A、B 和 C 做组装测试为止。这些工作步骤按图 9.15 来安排。在此图中，各边表示要完成的任务，边上均标注任务的名字，如“A 编码”表示模块 A 的编码工作。边上的数字表示完成该任务的持续时间。 图中有数字编号的结点是任务的起点和终点，在图中，0 号结点是整个任务网络的起点，8 号结点是终点。图中足够明确地表明了各项任务的计划时间，以及各项任务之间的依赖关系。

在组织较为复杂的项目任务时，或是需要对特定的任务进一步做更为详细的计划时，可以使用分层的任务网络图。

在软件工程项目中必须处理好进度与质量之间的关系。在软件开发实践中常常会遇到这样的事情，当任务未能按计划完成时，只好设法加快进度赶上去。但事实告诉我们，在进度压力下赶任务，其成果往往是以牺牲产品的质量为代价的。还应当注意到，产品的质量与生产率有着密切的关系。日本人有个说法：在价格和质量上折衷是不可能的，但高质量给生产者带来了成本的下降这一事实是可以理解的，这里的质量是指的软件工程过程的质量。

(5) 项目的追踪和控制

软件项目管理的一项重要工作就是在项目实施过程中进行追踪和控制。可以用不同的方式进行追踪：

§ 定期举行项目状态会议。在会上，每一位项目成员报告他的进展和遇到的问题。

§ 评价在软件工程过程中所产生的所有评审的结果。

§ 确定由项目的计划进度所安排的可能选择的正式的里程碑。

§ 比较在项目资源表中所列出的每一个项目任务的实际开始时间和计划开始时间。

§ 非正式地与开发人员交谈，以得到他们对开发进展和刚冒头的问题的客观评价。

实际上有经验的项目管理人员已经使用了所有这些追踪技术。软件项目管理人员还利用“控制”来管理项目资源、覆盖问题、及指导项目工作人员。如果事情进行得顺利(即项目按进度安排要求且在预算内实施，各种评审表明进展正常且正在逐步达到里程碑)，控制将是轻微的。但当问题出现的时候，项目管理人员必须实行控制以尽可能快地排解它们。在诊断出问题之后，在应用论域中可能需要一些追加资源；人员可能要重新部署，或者项目进度要重新调整。

9、软件项目的组织

(1) 项目任务的划分

软件项目的实施，如何进行工作的划分是实施计划首先应解决的问题。常用的计划结构有：

① 按阶段进行项目的计划工作

按软件生存期把全部项目开发工作划分为若干阶段(活动)，对每个阶段的工作做出计划。再把每个阶段的工作进一步分解为若干个任务，做出任务计划。还要把任务细分为若干步骤，做出步骤计划。这样三层次的计划成为整个项目计划的依据。显然，过细地做好分层计划，可以提高计划的精确度，减少或及早地发现问题。

② 任务分解结构

按项目本身的实际情况进行自顶向下的结构化分解，形成树形任务结构，如图 9.16 所示。进一步把工作内容、所需的工作量、预计完成的期限也规定下来。这样可以把划分後的工作落实到人，做到责任明确，便于监督检查。

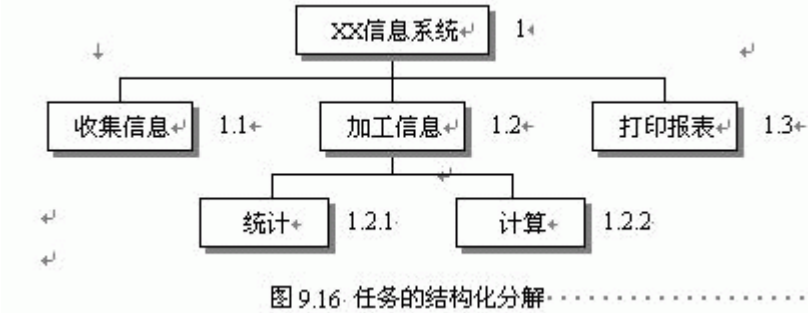


图 9.16 任务的结构化分解

③ 任务责任矩阵

在任务分解的基础上，把工作分配给相关人员，用一个矩阵形表格表示任务的分工和责任。例如，我们把图 9.16 已分解的任务分配给五位软件开发人员，表 9.17 表明了利用任务责任矩阵表达的分工情况。从图中可以看出，工作的责任和任务的层次关系都非常明确。

表 9.17 任务责任矩阵

编号	工作划分	负责人 张XX	系统工程师 王XX	系统工程师 李XX	程序员 赵XX	程序员 陈XX
1		审批				
1.1	收集信息		审查	设计	实现	
1.2	加工信息			审查		
1.2.1	统计		设计			实现
1.2.2	计算		设计			实现
1.3	打印报表		审查	设计	实现	

(2) 软件项目组织的建立

参加软件项目的人员组织起来，发挥最大的工作效率，对成功地完成软件项目极为重要。开发组织采用什么形式，要针对软件项目的特点来决定，同时也与参与人员的素质有关。人的因素是不容忽视的参数。

① 组织原则

在建立项目组织时应注意到以下原则：

§ 尽早落实责任：软件项目要尽早指定专人负责。使他有权有责。

§ 减少接口：一个组织的生产率是和完成任务中存在的通信路径数目成反比的。因此，要有合理的人员分工、好的组织结构、有效的通信，减少不必要的生产率的损失。

§ 责权均衡：软件经理人员所负的责任不应比委任给他的权力还大。

② 组织结构的模式

通常有三种组织结构的模式可供选择：

§ 按课题划分的模式

把软件人员按课题组成小组，小组成员自始至终参加所承担课题的各项任务，负责完成软件产品的定义、设计、实现、测试、复查、文档编制、甚至包括维护在内的全过程。

§ 按职能划分的模式

把参加开发项目的软件人员按任务的工作阶段划分成若干专业小组。待开发的软件产品在每个专业小组完成阶段加工(即工序)以后，沿工序流水线向下传递。例如，分别建立计划组、需求分析组、设计组、实现组、系统测试组、质量保证组、维护组等。各种文档按工序在各组之间传递。这种模式在小组之间的联系形成的接口较多，但便于软件人员熟悉小组的工作，进而变成这方面的专家。

各个小组的成员定期轮换有时是必要的，为的是减少每个软件人员因长期做单调的工作而产生的乏味感。

§ 矩阵形模式

这种模式实际上是以上两种模式的复合。一方面，按工作性质，成立一些专门组，如开发组、业务组、测试组等；另一方面，每一个项目又有它的经理人员负责管理。每个软件人员属于某一个专门组，又参加某一项目的工作。如图 9.17 所示。

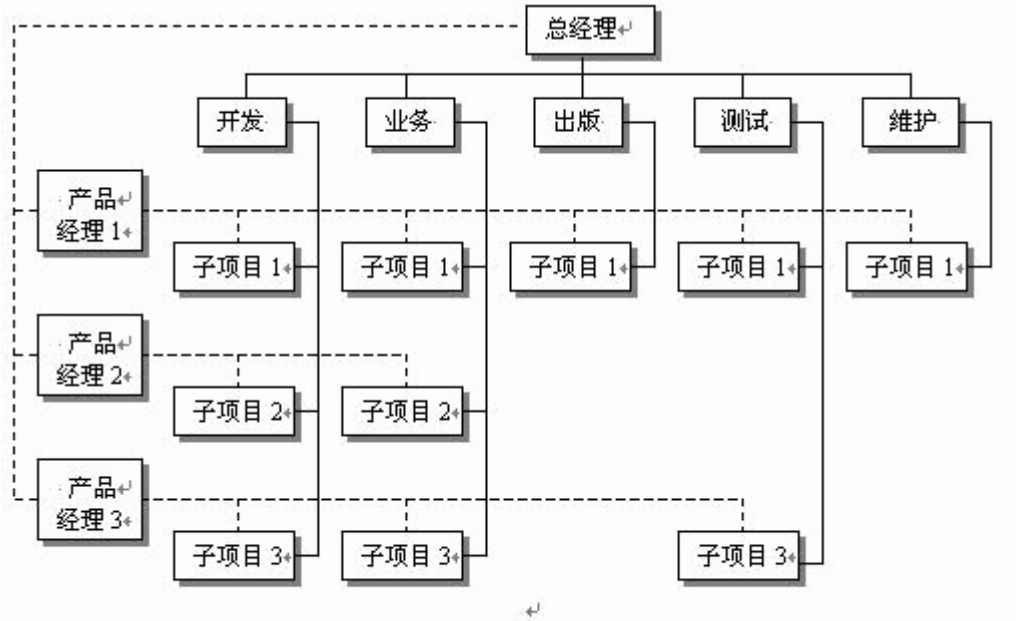


图 9.17 软件开发组织的矩阵形模式

矩阵形结构的组织具有一些优点：参加专门组的成员可在组内交流在各项目中取得的经验，这更有利于发挥专业人员的作用。另一方面，各个项目有专人负责，有利于软件项目的完成。显然，矩阵形结构是一种比较好的形式。

③ 程序设计小组的组织形式

通常认为程序设计工作是按独立方式进行的，程序人员独立地完成任务。但这并不意味着互相之间没有联系。人员之间联系的多少和联系的方式与生产率直接相关。程序设计小组内人数少，如 2～3 人，则人员之间的联系比较简单。但在增加人员数目时，相互之间的联系复杂起来，并且不是按线性关系增长。并且，已经进行中的软件项目在任务紧张，延误了进度的情况下，不鼓励增加新的人员给予协助。除非分配给新成员的工作是比较独立的任务，并不需要对原任务有更细致的了解，也没有技术细节的牵连。

小组内部人员的组织形式对生产率也有影响。现有的组织形式有三种。

§ 主程序员制小组

小组的核心由一位主程序员、二至五位技术员、一位后援工程师组成。主程序员负责小组全部技术活动的计划、协调与审查工作，还负责设计和实现项目中的关键部分。技术员负责项目的具体分析与开发，以及文档资料的编写工作。后援工程师支持主程序员的工作，为主程序员提供咨询，也做部分分析、设计和实现的工作。并在必要时能代替主程序员工作。主程序员制小组还可以由一些专家(例如通信专家或数据库设计专家(、辅助人员(例如，打字员和秘书(、软件资料员协助工作。

主程序员制的开发小组突出了主程序员的领导，参看图 9.18(a)。 这种集中领导的组织形式能否取得好的效果，很大程度上取决于主程序员的技术水平和管理才能。

§ 民主制小组

在民主制小组中，遇到问题，组内成员之间可以平等地交换意见，参见图 9.18(b)。这种组织形式强调发挥小组每个成员的积极性，要求每个成员充分发挥主动精神和协作精神。有人认为这种组织形式适合于研制时间长、开发难度大的项目。

§ 层次式小组

在层次式小组中，组内人员分为三级：组长(项目负责人)一人负责全组工作，包括任务分配、技术评审和走查、掌握工作量和参加技术活动。他直接领导二至三名高级程序员，每位高级程序员通过基层小组，管理若干位程序员。这种组织结构只允许必要的人际通信。比较适用于项目本身就是层次结构的课题。参看图 9.18(c)。

这种结构比较适合项目本身就是层次结构的课题。可以把项目按功能划分成若干子项目，把子项目分配给基层小组，由基层小组完成。基层小组的领导与项目负责人直接联系。这种组织方式比较适合于大型软件项目的开发。

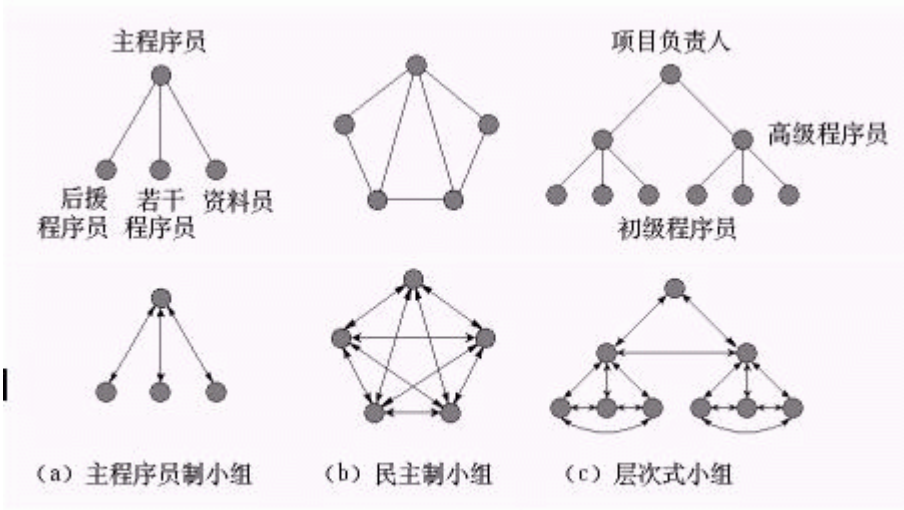


图 9.18 三种不同的小组结构
(上排的三种为结构形式，下排的三种为通信路径)

以上三种组织形式可以根据实际情况，组合起来灵活运用。总之，软件开发小组的主要目的是发挥集体的力量进行软件研制。

(3) 人员配备

如何合理地配备人员，也是成功地完成软件项目的切实保证。所谓合理地配备人员应包括：按不同阶段适时任用人员，恰当掌握用人标准。

① 项目开发各阶段所需人员

一个软件项目完成的快慢，取决于参与开发人员的多少。在开发的整个过程中，多数软件项目是以恒定人力配备的。如图 9.19 所示。

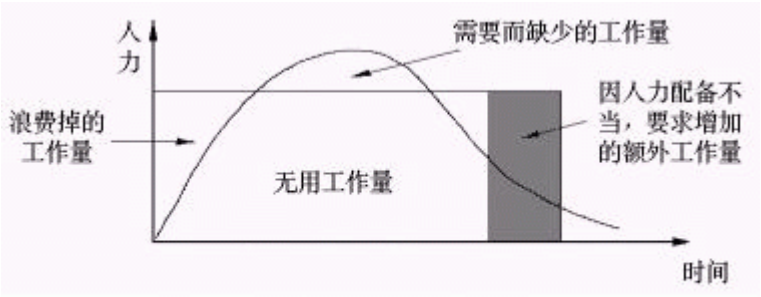


图 9.19 软件项目的恒定人力配备

按此曲线，需要的人力随开发的进展逐渐增加，在编码与单元测试阶段达到高峰，以后又逐渐减少。如果恒定地配备人力，在开发的初期，将会有部分人力资源用不上而浪费掉。在开发的中期(编码与单元测试)，需要的人力又不够，造成进度的延误。这样在开发的后期就需要增加人力以赶进度。因此，恒定地配备人力，对人力资源是比较大的浪费。

② 配备人员的原则

配备软件人员时，应注意以下三个主要原则：

§ 重质量——软件项目是技术性很强的工作，任用少量有实践经验、有能力的人员去完成关键性的任务，常常要比使用较多的经验不足的人员更有效。

§ 重培训——花力气培养所需的技术人员和管理人员是有效解决人员问题的好方法。

§ 双阶梯提升——人员的提升应分别按技术职务和管理职务进行，不能混在一起。

③ 对项目经理人员的要求

软件经理人员是工作的组织者，他的管理能力的强弱是项目成败的关键。除去一般的管理要求外，他应具有以下能力：

§ 把用户提出的非技术性的要求加以整理提炼，以技术说明书的形式转告给分析员和测试员。

§ 能说服用户放弃一些不切实际的要求，以便保证合理的要求得以满足。

§ 能够把表面上似乎无关的要求集中在一起，归结为“需要什么”，“要解决什么问题”。这是一种综合问题的能力。

§ 要懂得心理学，能说服上级领导和用户，让他们理解什么是不合理的要求。但又要使他们毫不勉强，乐于接受，并受到启发。

④ 评价人员的条件

软件项目中人的因素越来越受到重视。在评价和任用软件人员时，必须掌握一定的标准。人员素质的优劣常常影响到项目的成败。能否达到以下这些条件是不应忽视的。

§ 牢固掌握计算机软件的基本知识和技能。

§ 善于分析和综合问题，具有严密的逻辑思维能力。

§ 工作踏实、细致，不靠碰运气，遵循标准和规范，具有严格的科学作风。

§ 工作中表现出有耐心、有毅力、有责任心。

§ 善于听取别人的意见，善于与周围人员团结协作，建立良好的人际关系。

§ 具有良好的书面和口头表达能力。

(4) 指导与检验

指导的目的是在软件项目的过程中，动员和促进工作人员积极完成所分配的任务。实际上，指导也是属于人员管理的范围，是组织好软件项目不可缺少的工作。检验是软件管理的最后一个方面。它是对照计划检查执行情况的过程，同时也是对照软件工程标准检查实施情况的过程。在发现项目的实施与计划或标准有较大的偏离时，应采取措施加以解决。

① 指导工作的要点

在指导软件项目时需注意到以下几个方面的问题。

§ 鼓励——对工作的兴趣和取得显著成绩常常能够成为推动工作的积极因素。恰当而且及时地鼓励是非常重要的。

§ 引导——通常，人们愿意追随那些能够体谅个人要求或实际困难的领导。高明的领导人应能注意到这些，并能巧妙地把个人的要求和目标与项目工作的整体目标结合起来，至少应能做到在一定程度上的协调，而不应眼看着矛盾的存在和发展，以致影响工作的开展。从风险分析中可知，大幅度的人员调整是非常有害的，它会带来许多实际问题。即使是人员的临时观念也都要使项目付出不可见的代价，因而蒙受无形的损失。

§ 通信——在软件项目中充满了人际通信联系。必要的通信联络是不可少的。但实践表明，软件生产率是通信量的函数。如果人际通信数量过大，会使软件生产率迅速下降。

② 检验管理的要点

在检验管理时应当注意到以下问题：

§ 重大偏离——在软件项目实施过程中，必须注意发现工作的开展与已制定的计划之间、或与需遵循的标准(规范)之间的重大偏离。遇到有这种情况应及时向管理部门报告并采取相应的措施给予适当的处置。

§ 选定标准——检验管理需要事先确定应当遵循的标准(或规范)，使得软件项目的工作进展可以用某些客观、精确且有实际意义的标准加以衡量。

§ 特殊情况——任何事务在一般规律之外都会存在一些特殊情况。管理人员必须把注意力软件项目实施的一些特殊情况上，认真分析其中的一些特殊问题，加以解决。

③ 检验管理的工作范围

检验管理在软件项目中可能涉及到以下几个方面。

§ 质量管理——包括明确度量软件质量的因素和准则，决定质量管理的方法和工具，以及实施质量管理的组织形式。

§ 进度管理——检验进度计划执行的情况。

§ 成本管理——度量并控制软件项目的开销。

§ 文档管理——检验文档编写是否符合要求。

§ 配置管理——检验软件配置。

④ 软件项目中人的因素

软件产品是人们大量智力劳动的结晶，软件项目能否获得成功，人的因素在其中所起的作用比其它任何工程项目都突出。用户是否参与及软件人员的能力等人的因素对软件生产率的影响极大。在与软件成本相关的影响因素中，人员的能力是最大影响因素。如果在软件项目中能够充分发挥软件人员的积极性，使他们的才能得到尽量的施展，软件生产率(以单位时间内开发出的源程序的平均行数)最高可提高 4 倍多。

著名的软件工程专家 Tom DeMarco 积 30 年软件项目管理的经验，认为软件项目中对于人员的管理问题不能象其它事物那样简单地划分，机械地对待。他特别注意到项目的规模、成本、缺陷、加快开发的因素以及执行进度计划中的种种问题。积累了 500 个项目开发过程的数据，从中发现大约 15%的项目失败了。有的是一开始就被撤消，有的中途流产，有的推迟了进度，有的成果不能投入使用。而且项目规模越大，情况越糟。究其原因，绝大多数失败的项目竟找不出一个可以说得出口的技术障碍；而障碍却来自人员之间的联系问题、人员的任用问题、对上级或对雇主失望、工作缺乏动力或缺乏高额工程维持费用等等。这些人际关系问题的解决可归结于“软件项目社会学”。

关于软件人员的办公环境，有许多因素影响着软件工作的效率。DeMarco 曾于 1984 年到 1986 年在 62 个公司的 600 名软件人员中进行编码和测试的竞赛活动，并对竞赛结果进行统计分析。结果表明，除了对语言的熟悉程度、工作年限、工资收入等因素以外，环境因素因素着很大的作用。良好的办公环境可保证软件人员高质量地完成任务。这里所说的办公环境是指每个软件人员的办公室工作面积、办公环境安静程度、专用程度、电话干扰程度、工作时间内外面来访人员次数等。 DeMarco 说，你如果是一名项目管理人员，你为软件人员安排了任务，提供了工作条件，而对工作环境所带来的影响估计不足，你还是应该承担责任。

三、例题分析

【例 1】由于软件工程有如下的特点，使软件管理比其它工程的管理更为困难。软件产品(A)。(B)标准的过程。大型软件项目往往是(C)项目。(D)的作用是为有效地定量地进行管理，把握软件工程过程的实际情况和它所产生的产品质量。在制定计划时，应当对人力、项目持续时间、成本作出(E)；(H)实际

上就是贯穿于软件工程过程中一系列风险管理步骤。最后，每一个软件项目都要制定一个(F)，一旦(G)制定出来，就可以开始着手(H)。

供选择的答案：

A ~ C. ① 可见的 ② 不可见的 ③ “一次性” ④ “多次” ⑤ 存在 ⑥ 不存在

D ~ H. ① 进度安排 ② 度量 ③ 风险分析 ④ 估算

⑤ 追踪和控制 ⑥ 开发计划

答案：A. ②, B. ⑥, C. ③, D. ②, E. ④, F. ①, G. ⑥, H. ⑤。

分析：由于软件工程有如下的特点：软件产品不可见；不存在标准的软件过程；大型软件项目往往是“一次性”的项目，使得软件得管理比其它工程的管理更为困难；通常，软件人员和用户确定了软件项目的目标和范围之后，度量的作用就是为了有效地定量地进行管理。对开发过程进行度量的目的是为了改进开发过程，而对产品进行度量的目的是为了提高产品的质量。在软件项目管理过程中一个关键的活动是制定计划。在做计划时，必须就需要的人力、项目持续时间、成本作出估算；风险分析对于软件项目管理是决定性的，它实际上就是贯穿于软件工程过程中一系列风险管理步骤，其中包括风险识别、风险估计、风险评价和风险驾驭等步骤。每一个软件项目都要制定一个进度安排，但不是所有的进度都要一样地安排。一旦制定了开发计划，就可以开始着手追踪和控制活动。

【例 2】在软件项目估算时，将代码行 LOC 和功能点 FP 数据在两个方面使用：一是作为一个估算变量，度量软件每一个(A)的大小；一是联合使用从过去的项目中收集到的(B)和其它估算变量，进行成本和(C)估算。LOC 和 FP 是两种不同的估算技术，但两者有许多共同的特征，只是 LOC 和 FP 技术对于分解所需要的(D)不同。当用(E)作为估算变量时，功能分解是绝对必要且应达到很详细的程度，而用(F)作为估算变量时，分解程度可以不很详细。(E)是直接估算，(F)是间接估算。若计划人员对每个功能分别按最佳的、可能的、悲观的三种情况给出 LOC 或 FP 估计值，记作 a, m, b，则 LOC 或 FP 的期望值 E 的公式为(G)，m 是加权的最可能的估计值，遵循(H)。

供选择的答案：

A ~ C. ① 模块 ② 软件项目 ③ 分量 ④ 持续时间

⑤ 工作量 ⑥ 进度 ⑦ 基线数据 ⑧ 改进数据

D. ① 详细程度 ② 分解要求 ③ 改进过程 ④ 使用方法

E, F. ① FP ② LOC

G. → → ①. E=(a+m+b)/3 → → ②. E=(a+4m+b)/6
···· → ③. E=(2a+3m+4b)/3 → → ④. $E=\sqrt[3]{a \cdot 4m \cdot b}$
H. → ··①. χ概率 → ·②. γ概率 → → ③. β概率 → → ④. 泊松

答案：A. ③, B. ⑦, C. ⑤, D. ①, E. ②, F. ①, G. ②, H. ③。

分析：在软件项目估算时，将代码行 LOC 和功能点 FP 数据在两个方面使用：一是作为一个估算变量，度量软件每一个分量的大小；一是联合使用从过去的项目中收集到的基线数据(即对以往项目所做的估算值，保留下来作为后续项目的估算参考)和其它估算变量，进行成本和工作量估算。

LOC 和 FP 是两种不同的估算技术，但两者有许多共同的特征，项目计划人员首先给出一个有界的软件范围的叙述，再由此叙述尝试着把软件分解成一些小的可分别独立进行估算的子功能。然后对每一个子功能估算其 LOC 或 FP(即估算变量)。接着，把基线生产率度量(如，LOC / PM 或 FP / PM)用做特定的估算变量，导出子功能的成本或工作量。将子功能的估算进行综合后就能得到整个项目的总估算。

LOC 或 FP 估算技术对于分解所需要的详细程度是不同的。当用 LOC 做为估算变量时，功能分解是绝对必要的且需要达到很详细的程度。而估算功能点所需要的数据是宏观的量，当把 FP 当做估算变量时所需要的分解程度不很详细。我们还应注意，LOC 是直接估算的,而 FP 是通过估计输入、输出、数据文件、查询和外部接口的数目，以及 14 种复杂性校正值间接地确定的。避开所用到的估算变量，项目计划人员可对每一个分解的功能提出一个有代表性的估算值范围。利用历史数据或凭实际经验(当其它的方法失效时)，项目计划人员对每个功能分别按最佳的、可能的、悲观的三种情况给出 LOC 或 FP 估计值。记作 a、m、b。当这些值的范围被确定之后，也就隐含地指明了估计值的不确定程度。
接着计算 LOC 或 FP 的期望值 E。

E = (a + 4m + b) / 6. (加权平均)
其中，可能的估计值 m 是加权最重的最可能的估算值且遵循 β 概率分布。

【例 3】定义一个人参加劳动时间的长短为(A)，其度量单位为 PM(人月)或 PY(人年)。而定义完成一个软件项目(或软件任务)所需的(A)为(B)，其度量单位是人月 / 项目(任务)，记作 PM(人月)。进一步地，定义单位(A)所能完成的软件(C)的数量为软件(D)，其度量单位为 LOC / PM。它表明一般指(E)的一个平均值。例如，一个软件的开发工作量如下表所示。该软件共有源代码 2900 行，其中， 500 行用于测试，2400 行是执行(F)的源代码。则劳动生产率是(G)(LOC / PM)。

表 软件开发所需工作量例

阶 段	软件计划	需求分析	设 计	编 码	测 试	总 计
需要工作量(人月)	1.0	1.5	3.0	1.0	3.5	10.0

供选择的答案：

A, B, D. ① 生产率 ② 工作量 ③ 成本 ④ 劳动量

E. ① 开发全过程 ② 某开发阶段 ③ 软件生存期 ④ 某开发任务

F, C. ① 软件 ② 程序 ③ 进程 ④ 产品

G. ① 520 ② 120 ③ 320 ④ 240

答案：A. ④, B. ②, C. ④, D. ①, E. ①, F. ②, G. ④。

分析：定义一个人参加劳动时间的长短为劳动量，其度量单位为 PM(人月)，PY(人年)。它不同于工作量。而定义完成一个软件项目(或软件任务)所需的劳动量为工作量，其度量单位是人月／项目(任务)，记作 PM(人月)。进一步地，定义单位劳动量所能完成的软件产品的数量为软件生产率，其度量单位为 LOC／PM。它表明一般指开发全过程的一个平均值。题例所示的软件共有源代码 2900 行，其中，500 行用于测试，2400 行是执行程序的源代码。则劳动生产率是(2900－500)／10 ＝ 240（LOC／PM）。

【例 4】对于一个大型的软件项目，由于项目的复杂性，需要进行一系列的估算处理。主要按(A)和(B)手段进行。估算的方法分为三类：从项目的整体出发，进行(B)的方法称为(C)估算法。把待开发的软件细分，直到每一个子任务都已经明确所需要的开发工作量，然后把它们加起来，得到软件开发总工作量的方法称为(D)估算法。而把待开发的软件项目与过去已完成的软件项目做类比，区分出类似部分和不同部分分别处理的方法称为(E)估算法。(F)是由多位专家进行成本估算的方法。

供选择的答案：

A, B. ① 类推 ② 类比 ③ 分解 ④ 综合

C ~ F. ① 差别 ② 自顶向下 ③ 自底向上 ④ 专家判定技术

⑤ 循序渐进 ⑥ 比较

答案：A. ③, B. ①, C. ②, D. ③, E. ①, F. ④。

分析：对于一个大型的软件项目，由于项目的复杂性，开发成本的估算不是一件简单的事，要进行一系列的估算处理。主要靠分解和类推的手段进行。基本估算方法分为三类。

① 自顶向下的估算方法：这种方法的主要思想是从项目的整体出发，进行类推。即估算人员根据以前已完成项目所消耗的总成本(或总工作量)，来推算将要开发的软件的总成本(或总工作量)，然后按比例将它分配到各开发任务单元中去，再来检验它是否能满足要求。这种方法的优点是估算工作量小，速度快。缺点是对

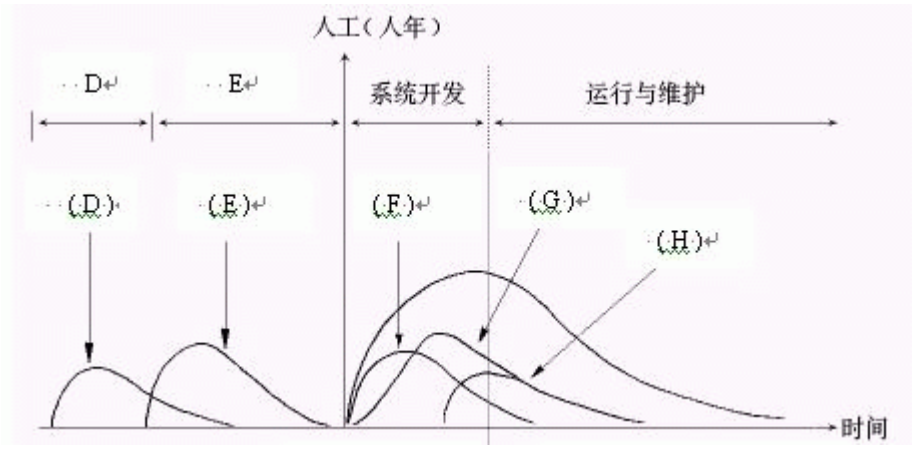
项目中的特殊困难估计不足，估算出来的成本盲目性大，有时会遗漏被开发软件的某些部分。

② 自底向上的估计法：这种方法的主要思想是把待开发的软件细分，直到每一个子任务都已经明确所需要的开发工作量，然后把它们加起来，得到软件开发的总工作量。这是一种常见的估算方法。它的优点是估算各个部分的准确性高。缺点是缺少各项子任务之间相互联系所需要的工作量，还缺少许多与软件开发有关的系统级工作量(配置管理、质量管理、项目管理)。所以往往估算值偏低，必须用其它方法进行检验和校正。

③ 差别估计法：这种方法综合了上述两种方法的优点，其主要思想是把待开发的软件项目与过去已完成的软件项目进行类比，从其开发的各个子任务中区分出类似的部分和不同的部分。类似的部分按实际量进行计算，不同的部分则采用相应的方法进行估算。这种的方法的优点是可以提高估算的准确程度，缺点是不容易明确“类似”的界限。

专家判定技术是由多位专家进行成本估算。由于单独一位专家可能会有种种偏见，最好由多位专家进行估算，取得多个估算值。

【例 5】 Putnam 模型是一种(A)模型。需要建立一条连续的(B)，称为 Rayleigh-Norden 曲线。可以由此导出一个(C)，把已交付的源代码(源语句)行数与工作量和开发时间联系起来。请选择合适的答案完成下面大型软件项目的开发工作量分布图。



供选择的答案：

A. ① 单值 ② 多值 ③ 静态多变量 ④ 动态多变量

⑤ 静态单变量

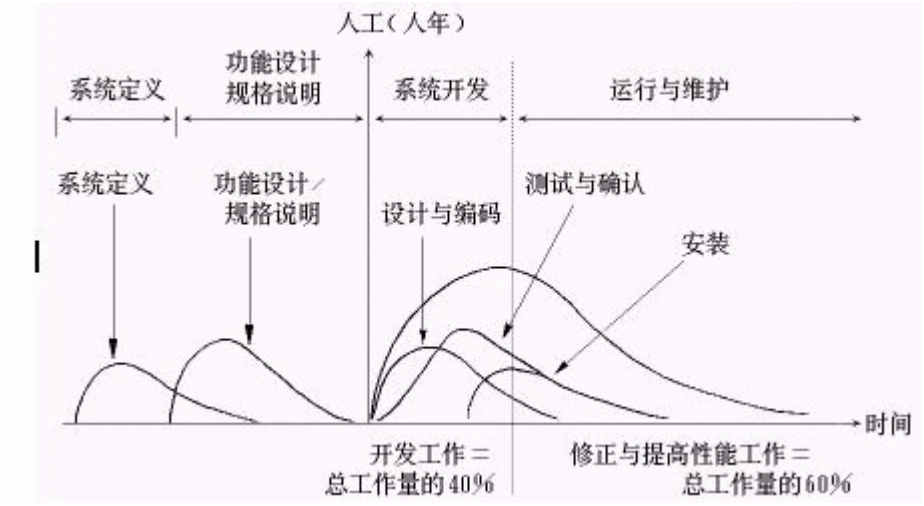
B, C. ① 软件方程 ② 函数曲线 ③ 资源需求曲线 ④ 工作量分布

D ~ H. ① 安装 ② 系统定义 ③ 测试与确认 ④ 设计与编码

⑤ 功能设计／规格说明

答案：A. ④, B. ④, C. ①, D. ②, E. ⑤, F. ④, G. ③, H. ①。

分析：Putnam 模型是 1978 年 Putnam 提出的模型，是一种动态多变量模型。它假定在软件开发的整个生存期中工作量有特定的分布。这种模型是依据在一些大型项目(总工作量达到或超过 30 个人年)中收集到的工作量分布情况而推导出来的，但也可以应用在一些较小的软件项目中。大型软件项目的开发工作量分布可以用下图所示的 Rayleigh-Norden 曲线表示。



该曲线的典型形状由 Lord Rayleigh 最早有分析地导出，并由 Norden 使用收集到的软件开发中的经验数据证实了这条曲线。用 Rayleigh-Norden 曲线可以导出一个“软件方程”，把已交付的源代码(源语句)行数与工作量和开发时间联系起来。

其中，td 是开发持续时间(以年计)， K 是软件开发与维护在内的整个生存期所花费的工作量(以人年计)，L 是源代码行数(以 LOC 计)，Ck 是技术状态常数，它反映出“妨碍程序员进展的限制”，并因开发环境而异。

【例 6】一个规模为 10KDSI 的商用微机远程通信的嵌入型软件，使用中间 COCOMO 模型进行软件成本估算。程序的名义工作量 MM = (A)；程序实际工作量 MM = (B)；开发所用的时间 TDEV = (C)；如果软件开发人员的工资都按每月 6000 美元计算，则该软件项目的开发人员的工资总额 = (D)。

表 1 中间 COCOMO 模型的名义工作量与进度公式

总体类型	工作量	进度
组织型	$MM=3.2(KDSI)^{1.05}$	$TDEV=2.5(MM)^{0.38}$
半独立型	$MM=3.0(KDSI)^{1.12}$	$TDEV=2.5(MM)^{0.35}$
嵌入型	$MM=2.8(KDSI)^{1.20}$	$TDEV=2.5(MM)^{0.32}$

表 2 影响工作量的因素 f_i 的取值

影响工作量因素 f_i	情况	取值
1 软件可靠性	只用于局部地区，恢复问题不严重	1.00（正常）
2 数据库规模	20000 字节	0.94（低）
3 产品复杂性	用于远程通信处理	1.30（很高）
4 时间限制	使用 70%的 CPU 时间	1.10（高）
5 存储限制	64K 中使用 45K	1.06（高）
6 机器	使用商用微处理机	1.00（额定值）
7 周转时间	平均 2 小时	1.00（额定值）
8 分析员能力	优秀人才	0.86（高）
9 工作经验	远程通信工作 3 年	1.10（低）
10 程序员能力	优秀人才	0.86（高）
11 工作经验	微型机工作 6 个月	1.00（正常）
12 语言使用经验	12 个月	1.00（正常）
13 使用现代程序设计技术	1 年以上	0.91（高）
14 使用软件工具	基本的微型机软件	1.10（低）
15 工期	9 个月	1.00（正常）

供选择的答案：

A, B. ① 45.8 ② 51.5 ③ 44.38 ④ 54.2

C. ① 8.9 月 ② 9.8 月 ③ 7.8 月 ④ 10.9 月

D. ① 26.4 万美元 ② 36 万美元 ③ 20.96 万美元 ④ 30.9 万美元

答案：A. ③, B. ②, C. ①, D. ④。

分析：考虑如题中表 2 的 15 种影响软件工作量的因素，通过定下乘法因子，修正 COCOMO 工作量公式和进度公式，可更合理地估算软件(各阶段)的工作量和进度。此时，实际工作量计算公式改成：

$$MM = r \times \prod_{i=1}^{15} f_i \times (KDSI)^c$$

由此得到程序名义工作量 $MM = 2.8*(10)^{1.20} = 44.38$ (MM)

程序实际工作量 $MM = 44.38 * \prod_{i=1}^{15} f_i = 44.38 * 1.17 = 51.5$ (MM)

开发所用时间 $TDEV = 2.5*(51.5)^{0.32} = 8.9$ (月)

如果分析员与程序员的工资都按每月 6,000 美元计算，则该项目的开发人员的工资总额为 51.5 * 6000 = 309000 （美元）

【例 7】在特定情况下，是否必须进行风险分析，是对项目开发的形势进行（A）后确定的。（ A ）可以按如下步骤进行：明确项目的目标、总策略、具体策略和为完成所标识的目标而使用的方法和资源；保证该目标是（ B ），项目成功的标准也是（ B ）；考虑采用某些条目作为项目成功的（ C ）；根据估计的结果来确定是否要进行风险分析。

一般来说，风险分析的方法要依赖于特定问题的需求和有关部门所关心的方面。具体分 3 步进行。第一步识别潜在的风险项，首先进行（ D ）过程；第二步估计每个风险的大小及其出现的可能性，选择一种（ E ），它可以估计各种风险项的值；第三步进行风险评估。风险评估也有三个步骤：确定（ F ），确定（ G ），把风险与“参照风险”做比较。

供选择的答案：

A. ① 风险管理 ② 风险估计 ③ 风险评价 ④ 风险测试

B. ① 可度量的 ② 不可度量的 ③ 准确的 ④ 不确定的

C. ① 规范 ② 标准 ③ 过程模型 ④ 设计要求

D, E. ① 信息分类 ② 信息收集 ③ 度量尺度 ④ 标准

⑤ 度量工具 ⑥ 信息获取

F, G. ① 风险的范围 ② 风险的特性 ③ 风险的级别 ④ 风险的评价标准

⑤ 风险的排除策略

答案：A. ②, B. ①, C. ②, D. ②, E. ③, F. ④, G. ③。
分析：在特定的情况下，是否必须进行风险分析，是对项目的开发形势进行风险估计后确定的。因为风险分析需要相当大的费用。只有在软件的费用、软件的作用、软件的性能及软件与系统的关系等各方面对系统有比较大的影响时，即软件的风险对于整个系统的成败，或对系统的风险有关键的影响时，才有必要进行软件的风险分析和管理。风险估计的步骤如下：

§ 明确项目的目标、总策略、具体策略和为完成所标识的目标而使用的方法和资源；

§ 保证该目标是可度量的，项目成功的标准也是可度量的；

§ 考虑采用以下的某些条目作为项目成功的标准：

① 最大限度的收益，② 最小的费用，③ 最小的风险损失，④ 最大限度的市场，⑤ 最小的周期性的波动，⑥ 形成有益的形象，⑦ 最佳的服务质量，⑧ 最高的增长率，⑨ 员工的满意度最高，⑩ 公司声望最高；

§ 根据估计的结果来确定是否要进行风险分析。

一般来说，风险分析的方法要依赖于特定问题的需求和有关部门所关心的方面。下面给出一种结构化的、一致的方法来进行风险分析。具体分 3 步进行。

第一步识别潜在的风险项。当确定要进行风险分析之后，就要收集信息，表明相关的风险。这就需要观察风险的征兆，理解其产生的原因，并列出所有的风险项。

首先进行信息收集。可以从过去完成的项目中收集已有的经验和收集来自群众的经验；可以模拟著名的事例；可以考虑类似的因素和进行常识性的判断；可以进行试验或测试以得到有关的结果，可以用各种方式来获得可能忽略的情况；此外，还可以针对经常发生的错误进行普查统计等。一般来说，通过过去的历史来认识软件项目的风险也许时一种最好的办法。例如，一些数字表明修复一个需求或设计阶段的错误的费用可能比修复一个测试阶段的错误的费用高 100 倍到 1000 倍。因此，可以把需求阶段标识为一个软件开发各阶段的风险区域。

然后进行信息分类。必须将收集到的信息以某种方式进行分类。一种简单而有用的方法是把风险项分为三类：有风险、可预见的风险、不可预见的风险。“有风险”是指经常发生的情况；“可预见的风险”是指以较高概率出现的情况；“不可预见的风险”是指不能识别的、未知的、不能观察的风险，是可能发生但事前很难预料的风险。对于每一种类型，还可以按其原因分成三种子类型：缺乏信息、缺乏管理及缺乏时间。其它分类方法可以按直接或间接分类，按运行性或策略性分类，按技术、进度、成本、支持分类。 第二步估计每个风险的大小及其出现的可能性，风险估计要度量所标识的各个风险可能造成的损失，即各种风险项的值(后果及程度)，用以减少度量的不确定性。可以按以下步骤进行：

选择某一种度量尺度，用以估计计算各种风险项的值，并具有合适的精度。由于要估计的风险信息可能有 3 种形式：叙述性、定性或定量，所以可选的尺度可以是命名尺度、序次尺度、坐标尺度或比例尺度。待估计的信息与度量尺度之间要建立对应关系，不同类型的信息有不同的度量尺度。例如，叙述性信息需要有命名尺度或序次尺度，定量的信息需要坐标或比例性的尺度。下表列出定量的风险等级：

风险等级	失效概率	说明
极高	0.99～0.81	超过当前的技术水平，肯定是技术问题。
很高	0.80～0.61	超过当前的技术水平，极像是技术问题。
高	0.60～0.50	最新的、尚未充分成熟的技术，好像是技术问题。
中等	0.49～0.24	最佳技术，只会有很小的技术问题。
低	0.24～0.10	使用的技术，没有技术问题。
很低	0.09～0.01	在使用中的系统。

在使用不同的方法和技术进行风险估计时常常会出现偏差，这是由于缺少可用来进行判断的信息，从而限制了风险估计的精度。由于信息分散，各人的理解和解释不同，造成“信息可用性偏差”。其次，选择的观念不同、专家的偏爱、采样规模的影响、样本相关的影响，以及修正的偏差等，都会产生估计的偏差。特别要注意的是：对于连续发生的事件和间断的不连续发生事件，这些偏差会造成什么样的影响。

必须采用一些技术来克服或消减风险估计中的不确定性。风险一般可以看成属于以下三种过程之一：行为型、自然型和随机型。

过程种类	克服和消减叙述型的不确定性	克服和消减度量型的不确定性
行为型	受定义人类行为的能力的限制	受理性的人类行为的限制
自然型	理论上无限制，但实际上有一定范围	受度量系统的精度的限制
随机型	理论上无限制，但实际上有一定范围	无法消减不确定性

第三步进行风险评估。因为软件项目所面临的是风险的一个较大的集合及其相互之间的影响，因此，必须针对这一点进行风险评估，以达到以下的目的：

首先，考虑各种风险的综合影响后，对已识别风险发生的可能性及其后果给出最终的量值(如果情况发生变化，也许要重新分析风险发生的可能性和可能的后果)；

其次，提供某种机制，对各个风险标明优先次序，以便予以适当安排；

最后，通过考虑其它可替代的方案，寻找避免风险的基本方法，即为高层决策人员提供全部必要的信息，以作出合理的有依据的决策。进行风险评估有三个步骤：

① 确定风险评估的标准。其目的是可用以衡量每个风险的后果，即判定在项目的生存期中各个阶段的风险的后果是否可以接受。此标准应与项目成功的标准相关。

② 确定风险的级别。其目的是把项目作为整体来评估。就是说，人们必须理解各种风险之间的相互作用，以及修改某些因素会如何影响它们之间的相互作用。为了说明可被评估的风险，引入“参照风险”。“参照风险”可以是一组单个风险的集合，或对项目会造成最大损害的一个或多个风险。必须仔细认清各风险间可能发生的耦合或复合情况。说明在把系统视为一个整体时，风险将导致系统失败的概率。

③ 把风险与“参照风险”做比较。把已评定的风险与在早期确定的“参照风险”相比较，结果可能是以下 3 种情况之一：

§ 可接受(评定的风险低于“参照风险”)；

§ 不可能接受(评定的风险大大高于“参照风险”)；

§ 不适合接受(评定的风险大于，但几乎等于“参照风险”)。

【例 8】假设开发某个计算机应用系统的投资额为 3000 元，该计算机应用系统投入使用后，每年可以节约 1000 元，5 年内可能节约 5000 元。3000 元是现在投资的钱，5000 元是 5 年内节省的钱，两者不能简单地比较。

假定年利率为 12%，利用计算货币现在价值的公式，可以算出该计算机应用系统投入使用后每年预计节省的金额的现在价值。

年	节省(元)	利率(1+0.12)ⁿ	现在价值(元)	累计现在价值(元)
1	1000	1.12	892.86	892.86
2	1000	1.25	800.00	1692.86
3	1000	1.40	714.29	2407.15
4	1000	1.57	636.94	3044.09
5	1000	1.76	568.18	3612.27

则该系统的纯收入是(A)，投资回收期是(B)，投资回收率为(C)。

供选择的答案：

A. ① 512.3 元 ② 729.28 元 ③ 602.4 元 ④ 612.27 元

B. ① 2. 4 年 ② 3.93 年 ③ 4.25 年 ④ 2.78 元

C. ① 25% ② 30% ③ 20% ④ 15%

答案：A. ④, B. ②, C. ③。

分析：效益包括经济效益，也包括社会效益。前者是有形的，后者是无形的。系统的经济效益等于因使用新系统而增加的收入加上使用新系统可以节省的运行费用。运行费用包括操作员人数、工作时间、消耗的物资等。在计算系统的经济效益时，应按照货币的时间价值来计算，这是因为对项目的投资在前，而系统效益的产生在后，且常常有一个较长的过程。 通常，用利率表示货币的时间价值。若设年利率为 i，现已存入 P 元，则 n 年后可得到的钱数为：F = (1 + i) n，F 就是 P 元钱在 n 年后的价值。 反之，若 n 年后能收入 F 元，那么这些钱现在的价值是：P = F / (1 + i) n。

由此，可从题意得： 该计算机应用系统在 5 年中的纯收入为：3612.27 - 3000 = 612.27 (元)。

投资回收期约为：3+(3000-2407.15)/(3044.09-2407.15)≈3.93(年)。

投资回收率设为 r，由下列方程式：

3000=1000/(1+r)+1000/(1+r)²+1000/(1+r)³+……+1000/(1+r)⁵

解得 r = 20%。

纯收入就是在整个生存期之内系统的累计经济效益(折合成现在值)与投资之差。投资回收期就是使累计的经济效益等于最初的投资所需的时间。投资回收率时投入资金所获得的利率。

【例 9】软件项目管理的主要职能包括：(A)，建立组织，配备人员，(B)和(C)。由于软件项目的特有性质，使得项目管理存在一定困难。第一、(D)，软件工程过程充满了大量高强度的脑力劳动；第二、(E)，在特定机型上，利用特定的硬件配置，由特定的系统软件和支撑软件支持，形成了特定的开发环境；第三、(F)，软件项目经历的各个阶段都深透了大量的手工劳动，远未达到自动化的程度；第四、(G)，用户要经过专门的培训，才能掌握操作步骤，且需要配备专职维护人员进行售后服务；第五、(H)，为高质量地完成软件项目，充分发掘人员的智力才能和创造精神。

在总结和分析足够数量失误的软件项目之后可知，造成软件失误的原因大多与(I)工作有关。在软件项目开始执行时，执行的过程中及项目进行的最后阶段都会遇到种种问题。

供选择的答案：

A ~ C. ① 编码 ② 制定计划 ③ 开发 ④ 指导

⑤ 测试 ⑥ 检验

D ~ H. ① 软件工作渗透了人的因素 ② 智力密集，可见性差

③ 单件生产 ④ 使用方法繁琐，维护困难

⑤ 劳动密集，自动化程度低

I. ① 设计 ② 维护 ③ 测试 ④ 管理

⑤ 实践 ⑥ 指导 ⑦ 审核 ⑧ 分析

答案：A. ②, B. ④, C. ⑥, D. ②, E. ③, F. ⑤, G. ④, H. ①, I. ④。

分析：软件管理的主要职能包括：

§ 制定计划：规定待完成的任务、要求、资源、人力和进度等。

§ 建立组织：为实施计划，保证任务的完成，需要建立分工明确的责任制机构。

§ 配备人员：任用各种层次的技术人员和管理人员。

§ 指导：鼓励和动员软件人员完成所分配的工作。

§ 检验：对照计划或标准，监督和检查实施的情况。

软件项目管理上的困难主要有：

① 智力密集，可见性差：软件工程过程充满了大量高强度的脑力劳动。软件开发的成果是不可见的逻辑实体，软件产品的质量难以用简单的尺度加以度量。对于不深入掌握软件知识或缺乏软件开发实践经验的人员，是不可能领导做好软件管理工作。软件开发任务完成得好也看不见，完成得不好有时也能制造假象，欺骗外行的领导。

② 单件生产：在特定机型上，利用特定的硬件配置，由特定的系统软件或支撑软件的支持，形成了特定的开发环境。再加上软件项目特定的目标，采用特定的开发方法、工具和语言，使得软件具有独一无二的特色，几乎找不到与之完全相同的软件产品。这种建立在内容、形式各异的基础上的研制或生产方式，与其它领域中大规模现代化生产有着很大的差别，也自然会给管理工作造成许多实际困难。

③ 劳动密集，自动化程度低：软件项目经历的各个阶段都渗透了大量的手工劳动，这些劳动又十分细致、复杂和容易出错。尽管近年来开展了软件工具和 CASE 的研究，但总体来说，仍远未达到自动化的程度。软件产业所处的这一状态，加上软件本身的复杂性，使得软件的开发和维护难于避免多种错误，软件的正确性难于保证，软件产品质量的提高自然受到了很大的影响。

④ 使用方法繁琐，维护困难：用户使用软件需要掌握计算机的基本知识，或者接受专门的培训，否则面对多种使用手册、说明和繁琐的操作步骤，学会使用要花费很大力气。另一方面，如果遇到软件运行出了问题，且没有配备专职维护人员，又得不到开发部门及时的售后服务，软件的使用者更是徒唤奈何。

⑤ 软件工作渗透了人的因素：为高质量地完成软件项目，充分发掘人员的智力才能和创造精神，不仅要求软件人员具有一定的技术水平和工作经验，而且还要求他们具有良好的心理素质。软件人员的情绪和他们的工作环境，对他们工作有很大的影响。与其它行业相比，它的这一特点十分突出，必须给予足够的重视。

造成软件失误的原因：

在总结和分析足够数量失误的软件项目之后，看出其原因大多与管理工作有关。在软件项目开始执行时，遇到的问题往往是：可供利用的资料太少；项目负责人的责任不明确；项目的定义模糊；没有计划或计划过分粗糙；资源要求未按时做出安排而落空；没有明确规定子项目完成的标准；缺乏使用工具的知识；项目已有更动，但预算未随之改变。

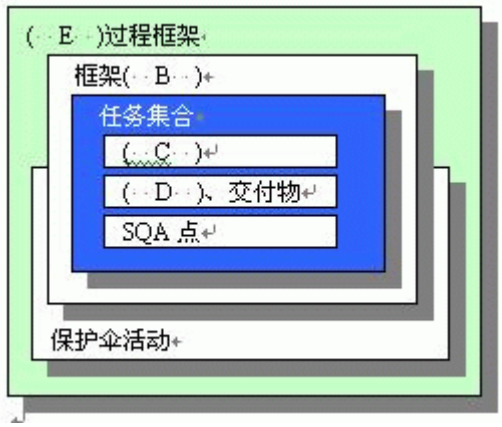
在软件项目执行的过程中可能会发生：项目审查只注意琐事而走过场；人员变动造成对工作的干扰；项目进行情况未能定期汇报；对阶段评审和评审中发现的问题如何处置未做出明确规定；资源要求并不像原来预计的那样大；未能做到严格遵循需求说明书；项目管理人员不足。 项目进行到最后阶段可能会发生：未做

质量评价；取得的知识和经验很少交流 ；未对人员工作情况做出评定；未做严格的移交；扩充性建议未写入文档资料。

总之，问题涉及到软件项目研制中的计划制定、进度估计、资源使用、人员配备、组织机构和管理方法等软件管理的许多侧面。

四、习题

【9-1】软件过程是软件（ A ）中的一系列相关软件工程（ B ）的集合。每一个软件过程又是由一组（C）、项目（D）、软件工程产品和交付物以及质量保证(SQA)点等组成。一个软件过程可以用右图的形式来表示。首先建立一个（ E ）过程框架，其中定义了少量可适用于所有软件项目的框架（ B ），再给出各个框架（ B ）的任务集合，最后是保护伞活动，如软件质量保证、软件配置管理以及测量等。软件过程模型的选择基于项目和应用的特点、采用的（ F ）和工具、要求的控制和需交付的产品。

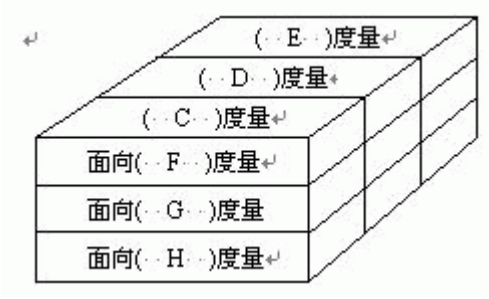


供选择的答案：

A ~ F. ① 工程 ② 公共 ③ 活动 ④ 生存期

⑤ 方法 ⑥ 工作任务 ⑦ 功能 ⑧ 里程碑

【9-2】软件的度量包括（ A ）和（ B ）。软件产品的（ A ）包括产生的代码行数、执行速度等。软件产品的（ B ）则包括若干质量特性。我们还可进一步将软件度量如右图所示那样分类。软件（ C ）度量主要关注软件工程过程的结果；（ D ）度量则指明了软件适应明确和不明确的用户要求到什么程度；（ E ）度量主要关注软件的一些特性而不是软件开发的全过程。从图中还可看到另一种分类方法：面向（ F ）的度量用于收集与直接度量有关软件工程输出的信息和质量信息。面向（ G ）的度量提供直接度量的尺度。面向（ H ）的度量则收集有关人们开发软件所用方式的信息和人们理解有关工具和方法的效率的信息。



供选择的答案：

A ~ B. ① 直接度量 ② 尺度度量 ③ 二元度量 ④ 间接度量

C ~ E. ① 质量 ② 技术 ③ 成本 ④ 生产率

F ~ H. ① 过程 ② 对象 ③ 人 ④ 存取

⑤ 规模 ⑥ 进程 ⑦ 功能 ⑧ 数据

【9-3】估算资源、成本和进度时需要经验、有用的历史信息、足够的定量数据和作定量度量的勇气。通常估算本身带有(A)。项目的复杂性越高，规模越大，开发工作量(B)，估算的(A)就(C)。项目的结构化程度提高，进行精确估算的能力就能(D)，而风险将(E)。有用的历史信息(F)，总的风险会减少。

供选择的答案：

A. ① 风范（范型） ② 风格 ③ 风险 ④ 度量

B ~ F. ① 增加 ② 越多 ③ 降低 ④ 不变

⑤ 越少 ⑥ 越高 ⑦ 越大

【9-4】 在考虑各种软件开发资源时，(A)是最重要的资源。如果把软件开发所需的资源画成一个金字塔形:在塔的上层是最基本的资源(A),在底部为(B)。(B)包括硬件资源和软件资源。(C)、(D)和其它硬件设备属于硬件资源。IPSE 工具属于软件资源中的(E)。为了提高软件的生产率和软件产品的质量，可建立(F)。

供选择的答案：

A, B. ① 方法 ② 人力 ③ 工具 ④ 上下文环境

C, D. ① 虚拟机 ② 目标机 ③ 自动机 ④ 宿主机

E, F. ① 维护工具 ② 分析设计工具 ③ 支持工具 ④ 编程工具

⑤ 可复用构件库 ⑥ 框架工具 ⑦ 原型化模拟工具

【9-5】任何软件项目都必须做好项目管理工作，最常使用的进度管理工具是（ A ），当某一开发项目的进度有可能拖延时，应该（ B ）。对于一个典型的软件开发项目，各开发阶段需投入的工作量的百分比大致是（ C ）。各阶段所需不同层次的技术人员大致是（ D ），而管理人员在各阶段所需数量也不同，相对而言大致是（ E ）。

供选择的答案：

A. ① 数据流图 ② 程序结构图 ③ 因果图 ④ PERT 图

B. ① 增加新的开发人员 ② 分析拖期原因加以补救

③ 从别的小组抽调人员临时帮忙 ④ 推迟预定完成时间

			需求分析	设 计	编 码	测 试
C.	投入工作量	①	25	25	25	25
		②	10	20	30	40
		③	15	30	15	40
		④	5	10	65	30
D.	技术人员水平	①	初级	高级	高级	高级
		②	中级	中级	高级	中级
		③	高级	中高级	初级	中高级
		④	中级	中高级	中级	初级
E.	管理人员数量	①	多	中	少	中
		②	中	中	中	中
		③	多	少	多	多
		④	少	多	少	多

【9-6】一个 32KDSI 的声音输入系统是一个输入原型，或是一个可行性表演模型。所需可靠性非常低，因为它不打算投入生产性使用。把此模型看做半独立型软件。试问该软件的名义工作量和实际工作量。

【9-7】风险分析实际上是 4 个不同的活动，按顺序依次为（ A ）、（ B ）、风险评价和（ C ）。在风险评价时，应当建立一个三元组：[ri, li, xi]，ri 是风险描述，li 是（ D ），而 xi 是风险的影响。一个对风险评价很有用的技术是定义（ E ）。（ F ）、（ G ）、（ H ）是三种典型的（ E ）。在做风险分析的上下文环境中一个（ E ）就存在一个单独的点，叫做参照点或（ I ）。在这个点上要公正底给出判断。实际上，参照点能在图上表示成一条平滑的曲线的情况很少，多数情况它是一个（ J ）。

供选择的答案：

A ~ C. ① 风险驾驭和监控 ② 风险识别 ③ 风险估计 ④ 风险消除
D. ① 风险的大小 ② 风险的概率 ③ 风险的时间 ④ 风险的范围

E. ① 风险参照水准 ② 风险度量 ③ 风险监控 ④ 风险工具

F ~ H. ① 生产率 ② 功能 ③ 成本 ④ 进度

⑤ 范围 ⑥ 性能

I, J. ① 凹点 ② 崩溃点 ③ 终点 ④ 区域

⑤ 拐点 ⑥ 原点

【9-8】对于一个小型的软件开发项目，一个人就可以完成需求分析、设计、编码和测试工作。但随着软件项目规模增大，需要有多人共同参与同一软件项目的工作。当几个人共同承担软件开发项目中的某一任务时，人与人之间必须通过交流来解决各自承担任务之间的(A)问题，即通信问题。通信需花费时间和代价，会引起软件错误(B)，(C)软件生产率。如果一个软件开发小组有 n 个人，每两人之间都需要通信，则共有(D)条通信路径。假设一个人单独开发软件，生产率是 5000 行/人年，且在每条通信路径上耗费的工作量是 250 行/人年。若 4 个人组成一个小组共同开发这个软件，则小组中每个人的软件生产率为(E)。若小组有 6 名成员，则小组中每个成员的软件生产率为(F)。因此，有人提出，软件开发小组的规模不能太大，人数不能太多，一般在(G)人左右为宜。 供选择的答案：

A. ① 分配 ② 管理 ③ 接口 ④ 协作

B, C. ① 降低 ② 增加 ③ 不变

D. ① n(n+1)/2 ② n(n-1)/2 ③ n(n-1)(n-2)/6 ④ n2/2

E, F. ① 4875 ② 4375 ③ 4625 ④ 5735

G. ① 8～15 ② 1～2 ③ 2～5 ④ 2～8

【9-9】软件项目的进度管理有许多方法，但(A)不是常用的进度控制图示方法。在几种进度控制图示方法中，(B)难以表达多个子任务之间的逻辑关系，使用(C)不仅能表达子任务之间的逻辑关系，而且可以找出关键子任务。在(C)中，用带箭头的边表示(D)，用圆圈结点表示(E)，它标明(D)的(F)。

供选择的答案：

A ~ C. ① 甘特图 ② IPO ③ PERT ④ 时标网状图

D ~ F. ① 数据流 ② 控制流 ③ 事件 ④ 处理

⑤ 起点或终点 ⑥ 任务

【9-10】软件项目组织的原则是(A)、(B)和(C)。一般有(D)、(E)、(F)三种组织结构的模式。(F)实际上是(D)和(E)两种模式的复合。(E)这种模式在小组之间的联系形成的接口较多，但便于软件人员熟悉小组的工作，进而成为这方面的专家。

供选择的答案：

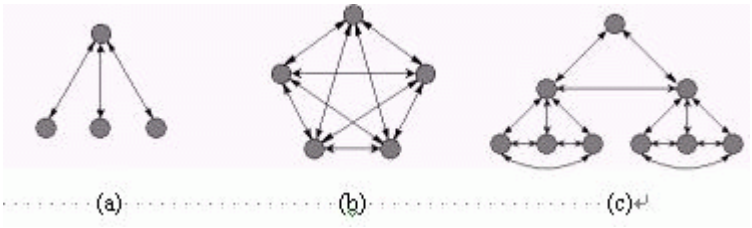
A ~ C. ① 推迟责任的落实 ② 尽早落实责任 ③ 减少接口

④ 增加联系 ⑤ 责权分离 ⑥ 责权均衡

D ~ F. ① 矩阵形模式 ② 主程序员小组模式 ③ 按课题划分的模式

④ 按职能划分的模式 ⑤ 民主制小组模式

【9-11】软件开发小组的目的是发挥集体的力量进行软件研制。因此，小组从培养(A)的观点出发进行程序设计消除软件的(B)的性质。通常，程序设计小组的组织形式有三种，如下图所示的a属于(C)，b属于(D)，c属于(E)。



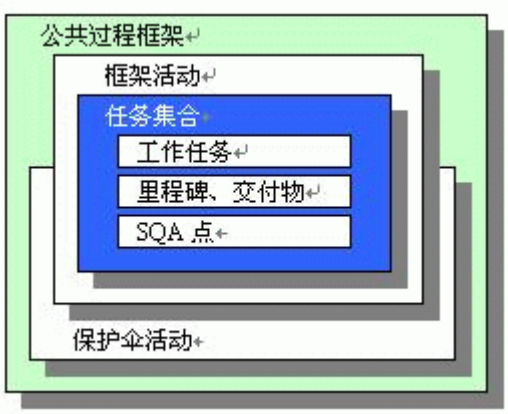
供选择的答案：

A, B. ① “局部” ② “全局” ③ “集体” ④ “个人”

C ~ E. ① 层次式小组 ② 民主制小组 ③ 主程序员制小组

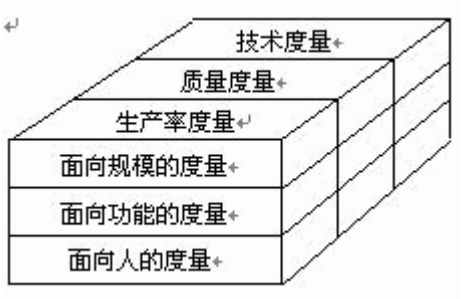
五、习题解答

【9-1】A. ④, B. ③, C. ⑥, D. ⑧, E. ②, F. ⑤。



软件过程是软件生存期中的一系列相关软件工程活动的集合。每一个软件过程又是由一组工作任务、项目里程碑、软件工程产品和交付物以及质量保证(SQA)点等组成。一个软件过程可以用右图的形式来表示。首先建立一个公共过程框架，其中定义了少量可适用于所有软件项目的框架活动，而不考虑它们的规模和复杂性。再给出各个框架活动的任务集合，使得框架活动能够适合于项目的特点和项目组的需求。最后是保护伞活动，如软件质量保证、软件配置管理以及测量等，它们独立于任何一个框架活动并将贯穿于整个过程。软件过程模型的选择基于项目和应用的特点、采用的方法和工具、要求的控制和需交付的产品。

【9-2】A. ①, B. ④, C. ④, D. ①, E. ②, F. ⑤, G. ⑦, H. ③。



软件的度量包括直接度量和间接度量。软件产品的直接度量包括产生的代码行数、执行速度、存储量大小、在某种时间周期中所报告的差错数。软件产品的间接度量则包括功能性、复杂性、效率、可靠性、可维护性和许多其它的质量特性。只要事先建立特定的度量规程，很容易做到直接度量开发软件所产生的代码行数等。但是，软件的功能性、效率、可维护性等质量特性却很难用直接度量判明，只有通过间接度量才能推断。我们还可进一步将软件度量如图所示那样分类。软件生产率度量主要关注软件工程过程的结果；软件质量度量则指明了软件适应明确和不明确的用户要求(软件使用合理性)到什么程度；技术度量主要关注软件的一些特性(如逻辑复杂性、模块化程度)而不是软件开发的全过程。从图中还可以看到另一种分类方法：面向规模的度量用于收集与直接度量有关的软件工程输出的信息和质量信息。面向功能的度量提供直接度量的尺度。面向人的度量则收集有关人们开发软件所用方式的信息和人们理解有关工具和方法的效率的信息。

估算资源、成本和进度时需要经验、有用的历史信息、足够的定量数据和作定量度量的勇气。通常估算本身带有（ A ）。项目的复杂性越高，规模越大，开发工作量（ C ），估算的（ A ）就（ D ）。项目的结构化程度的提高，进行精确估算的能力就能（ E ），而风险将（ F ）。有用的历史信息（ G ），总的风险会减少。

供选择的答案：

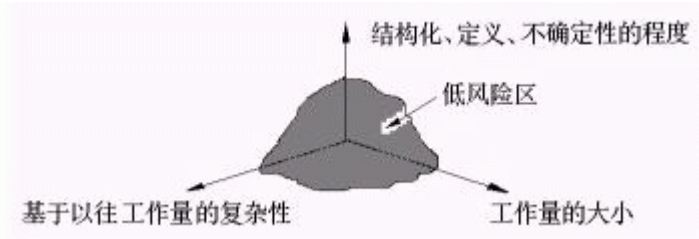
A. ①风范（范型） ② 风格 ③ 风险 ④ 度量

B ~ G. ① 增加 ② 越大 ③ 降低 ④ 不变

⑤ 减少 ⑥ 越高

【9-3】A. ③, B. ②, C. ⑦, D. ①, E. ③, F. ②。

估算资源、成本和进度时需要经验、有用的历史信息、足够的定量数据和作定量度量的勇气。估算本身带有风险。增加风险的各种因素如图所示。



项目的复杂性对于增加软件估算的不确定性影响很大。复杂性越高，估算的风险就越高。但是，复杂性是相对度量，它与项目参加人员的经验有关。

项目的规模对于软件估算的精确性和功效影响也比较大。因为随着软件规模的扩大，软件元素之间的相互依赖、相互影响程度迅速增加，因而估算的一个重要方法——问题分解会变得更加困难。由此可知，项目的规模越大，开发工作量越大，估算的风险越高。 项目的结构化程度也影响项目估算的风险。所谓结构性是指功能分解的简便性和处理信息的层次性。结构化程度的提高，进行精确估算的能力就能提高，而风险将减少。

历史信息的有效性也影响估算的风险。回顾过去，就能够仿效做过的事，且改进出现问题的地方。在对过去的项目进行综合的软件度量之后，就可以借用来比较准确地进行估算，安排进度以避免重走过去的弯路，而总的风险也减少了。

风险靠对不确定性程度定量地进行估算来度量，此外，如果对软件项目的作用范围还不十分清楚，或者用户的要求经常变更，都会导致对软件项目所需资源、成本、进度的估算频频变动，增加估算的风险。

【9-4】A. ②, B. ③, C. ②, D. ④, E. ⑥, F. ⑤。其中，C、D 的答案顺序可互换。



软件项目计划的第二个任务是对完成该软件项目所需的资源进行估算。若把软件开发所需的资源画成一个金字塔，在塔的底部必须有现成的用以支持软件开发的工具——硬件工具及软件工具，在塔的高层是最基本的资源——人。在考虑各种软件开发资源时，人是最重要的资源。在安排开发活动时必须考虑人员的技术水平、专业、人数、以及在开发过程各阶段中对各种人员的需要。

硬件是作为软件开发项目的一种工具而投入的。在软件项目计划期间，考虑三种硬件资源：宿主机(软件开发时使用的计算机及外围设备)；目标机(运行已开发成功软件的计算机及外围设备)；其它硬件设备(专用软件开发时需要的特殊硬件资源)。宿主机连同必要的软件工具构成一个软件开发系统。通常这样的开发系统能够支持多种用户的需要，且能保持大量的由软件开发小组成员共享的信息。但在许多情况下，除了那些很大的系统之外，不一定非要配备专门的开发系统。因此，所谓硬件资源，可以认为是对现存计算机系统的使用，而不是去购买一台新的计算机。宿主机与目标机可以是同一种机型。

软件在开发期间使用了许多软件工具来帮助软件的开发。这些软件工具叫做计算机辅助软件工程(CASE)。主要的软件工具分类为：业务系统计划工具集；项目管理工具集；支持工具；分析和设计工具；编程工具；组装和测试工具；原型化和模拟工具；维护工具；框架工具。这些框架工具能够提供一个建立集成项目支撑环境(IPSE)的框架。在多数情况，框架工具实际提供了数据库管理和配置管理的能力与一些实用工具，能够把各种工具集成到 IPSE 中。

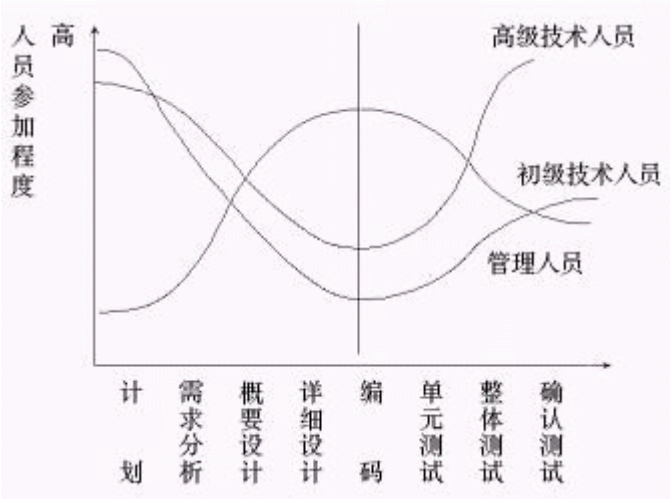
为了促成软件的复用，以提高软件的生产率和软件产品的质量，可建立可复用的软件构件库。根据需要，对软件构件稍做加工，就可以构成一些大的软件包。这要求这些软件构件应加以编目，以利引用，并进行标准化和确认，以利于应用和集成。

【9-5】A. ④, B. ②, C. ③, D. ③, E. ①。

PERT 技术叫做计划评审技术，是安排开发进度，制定软件开发计划的最常用的方法。它采用网络图来描述一个项目的任务网络。通常用两张表来定义网络图。一张表给出与一特定软件项目有关的所有任务(也称为任务分解结构)，另一张表给出应当按照什么样的次序来完成这些任务(有时称为限制表)。

当某一开发项目的进度有可能拖延时，应该分析拖期原因加以补救，切忌中途加人，否则反而会降低软件生产率。对于一个典型的软件开发项目，各开发阶段需投入的工作量的百分比大致遵循 40－20－40 规则。即在整个软件开发过程中，编码的工作量占 20%，编码前的工作量占 40%，编码后的工作量占 40%。

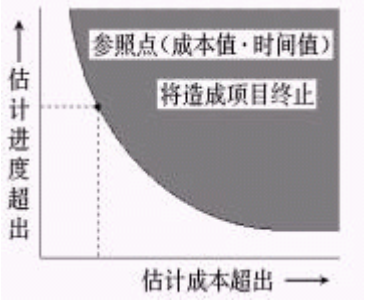
对于一些规模较小的项目(1 个人年或者更少)，只要向专家做些咨询，也许一个人就可以完成所有的软件工程步骤。而对一些规模较大的项目，在整个软件生存期中，各种人员的参与情况是不一样的。如图所示。在软件计划和需求分析阶段，对软件系统进行定义，主要工作是由管理人员和高级技术人员在做，初级技术人员参与较少。待到对软件进行具体设计、编码及测试时，管理人员逐渐减少对开发工作的参与，高级技术人员主要在设计方面把关，具体编码及调试参与较少，大量的工作将由初级技术人员去做。到了软件开发的后期，需要对软件进行检验、评价和验收，管理人员和高级技术人员又将投入很多的精力。



【9-6】 对于这样一个规模为 10KDSI 的商用微机远程通信的嵌入式软件，使用中间 COCOMO 模型进行软件成本估算。名义工作量为 MM = 3.0*(10)1.12 = 146 (人月)。又查表知 f1 = 0.75，其它 fi = 1.00，则最终计算出的实际工作量为 MM = 146 * 0.75 = 110 (人月)。

【9-7】A. ②, B. ③, C. ①, D. ②, E. ①, F. ③, G. ④, H. ⑥, I. ②, J. ④。其中，F、G、H 的答案顺序可互换。

风险分析实际上是 4 个不同的活动： 风险识别，风险估计，风险评价和风险驾驭与监控。 在进行风险评价时，可建立一系列三元组：[ri, li, xi]，其中，ri 是风险，li 是风险出现的可能性(概率)，而 xi 是风险产生的影响。在做风险评价时，应进一步审查在风险估计时所得到的估计的准确性，尝试对已发现的风险进行优先排队，并着手考虑控制和／或消除可能出现风险的方法。



在做风险评价时常采用的一个非常有效的方法就是定义风险参照水准。对于大多数软件项目来说，性能、支持、成本、进度就是典型的风险参照水准。就是说，对于成本超支、进度延期、性能降低、支持困难，或它们的某种组合，都有一个水准值，超出它就会导致项目被迫终止。在软件风险分析的上下文中，一个风险参照水准就有一个点，叫做参照点或崩溃点。在这个点上，要公平地给出可接受的判断，看是继续执行项目工作，还是终止它们（出的问题太大）。实际上，参照点能在图上被表示成一条平滑的曲线的情况很少。在多数情况中，它是一个区域，在此区域中存在许多不确定性的范围。

【9-8】A. ③, B. ②, C. ①, D. ②, E. ③, F. ②, G. ④。

对于一个小型的软件开发项目，一个人就可以完成需求分析、设计、编码和测试工作。但是，随着软件开发项目规模的增大，就会有更多的人共同参与同一软件项目的工作。例如 10 个人 1 年可以完成的项目，若让 1 个人干 10 年是不行的。因此，需要多人组成开发小组共同参加一个项目的开发。但是，当几个人共同承担软件开发项目中的某一任务时，人与人之间必须通过交流来解决各自承担任务之间的接口问题，即所谓通信问题。通信需花费时间和代价，会引起软件错误增加，降低软件生产率。

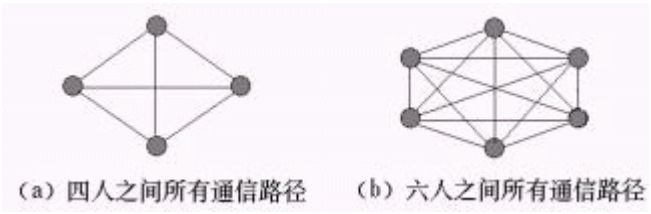
若两个人之间需要通信，则称在这两个人之间存在一条通信路径。如果一个软件开发小组有 n 个人，每两人之间都需要通信，则总的通信路径有 n(n-1)/2 条。

假设一个人单独开发软件，生产率是 5000 行 / 人年。若 4 个人组成一个小组共同开发这个软件，则需要 6 条通信路径(图(a))。若在每条通信路径上耗费的工作量是 250 行 / 人年。则小组中每个人的软件生产率降低为

5000－6×250 / 4 ＝ 5000－375 ＝ 4625 行 / 人年。

如果小组有 6 名成员,通信路径增加到 15 条(图(b))。每条通信路径消耗的工作量不变，则小组中每个成员的软件生产率降低为

5000－15×250 / 6 ＝ 5000－625 ＝ 4375 行 / 人年。

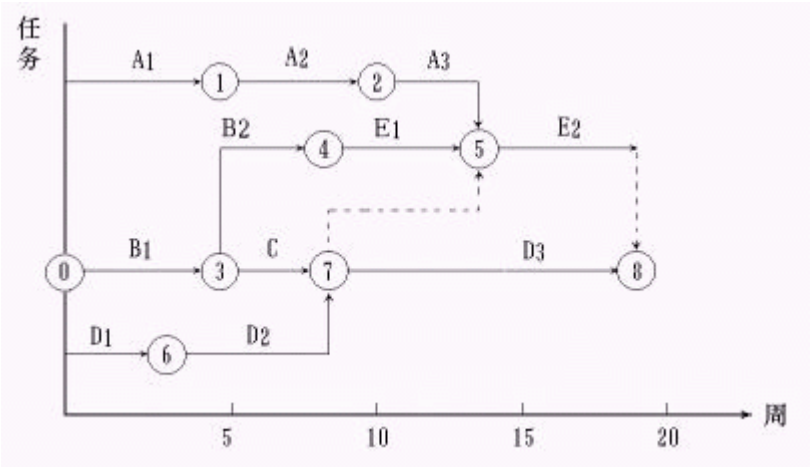


从上述简单分析可知，一个软件任务由一个人单独开发，生产率最高；而对于一个稍大型的软件项目，一个人单独开发，时间太长。因此软件开发小组是必要的。有人提出，软件开发小组的规模不能太大，人数不能太多，一般在 2～8 人左右为宜。

【9-9】A. ②, B. ①, C. ③, D. ⑥, E. ③, F. ⑤。

软件项目的进度计划和工作的实际进展情况，对于较大的项目来说，难以用语言叙述清楚。特别是表现各项任务之间进度的相互依赖关系，需要采用图示的方法。常用的图示方法有甘特图、时标网状图、PERT 等，IPO 图是用于在结构化设计中描述程序结构中输入－处理－输出的，不是进度控制的图示工具。 甘特图以水平线段表示任务的工作阶段；线段的起点和终点分别对应着任务的开工时间和完成时间；线段的长度表示完成任务所需的时间。从甘特图上可以很清楚地看出各子任务在时间上的对比关系，并以文档编制与评审作为软件开发进度的里程碑。甘特图的优点是标明了各任务的计划进度和当前进度，能动态地反映软件开发进展情况。缺点是难以反映多个任务之间存在的复杂的逻辑关系。

时标网状图克服了甘特图的缺点，用具有时标的网状图来表示各个任务的分解情况，以及各个子任务之间在进度上的逻辑依赖关系(参看下图)。时标网状图中的箭头(直线、折线)表示各任务间的(先决)依赖关系；箭头上的名字表示任务代号；箭头的水平长度表示完成该任务的时间；而圆圈表示一个任务结束、另一个任务开始的事件。



PERT 图也叫做计划评审技术，它采用网络图来描述一个项目的任务网络。不仅可以表达子任务的计划安排，还可以在任务计划执行过程中估计任务完成的情况，分析某些子任务完成情况对全局的影响，找出影响全局的区域和关键子任务，以便及时采取措施，确保整个项目的完成。在 PERT 图中，用箭头表示任务或子任务，箭头上附带的数字表示完成任务所需的时间；圆形结点表示事件，每一事件标明某些任务都已完成，下面另外一些任务将要开始。

【9-10】A. ②, B. ③, C. ⑥, D. ③, E. ④, F. ①。其中，A、B、C 答案顺序可互换。

在建立项目组织时应注意到以下原则：

§ 尽早落实责任：在软件项目工作的开始，要尽早指定专人负责。使他有权进行管理，并对任务的完成负全责。

§ 减少接口：在开发过程中，人与人之间的联系是必不可少的，存在着通信路径。一个组织的生产率是和完成任务中存在的通信路径数目是相互抵触的。 因此，要有合理的人员分工、好的组织结构、有效的通信，减少不必要的生产率的损失。

§ 责权均衡：软件经理人员所负的责任不应比委任给他的权力还大。 通常有三种组织结构的模式可供选择：

§ 按课题划分的模式：把软件开发人员按课题组成小组，小组成员自始至终参加所承担课题的各项任务。他们应负责完成软件产品的定义、设计、实现、测试、复查、文档编制、甚至包括维护在内的全过程。

§ 按职能划分的模式：把参加开发项目的软件人员按任务的工作阶段划分成若干个专业小组。要开发的软件产品在每个专业小组完成阶段加工（即工序）以后，沿工序流水线向下传递。例如，分别建立计划组、需求分析组、设计组、实现组、系统测试组、质量保证组、维护组等。各种文档资料按工序在各组之间传递。这种模式在小组之间的联系形成的接口较多，但便于软件人员熟悉小组的工作，进而变成这方面的专家。 § 矩阵形模式：这种模式实际上是以上两种模式的复合。一方面，按工作性质，成立一些专门组，如开发组、业务组、测试组等；另一方面，每一个项目又有它的经理人员负责管理。每个软件人员属于某一个专门组，又参加某一项目的工作。

【9-11】A. ②, B. ④, C. ③, D. ②, E. ①。

软件开发小组的主要目的是发挥集体的力量进行软件研制。因此，小组培养从“全局”的观点出发进行程序设计，消除软件的“个人”性质，并促进更充分的复审，小组提倡在共同工作中互相学习从而改善软件的质量。 小组内部人员的组织形式对生产率也有影响。现有的组织形式有三种。

① 主程序员制小组：突出了主程序员的领导，强调主程序员与其他技术人员的直接联系，简化了人际通信。这种集中领导的组织形式能否取得好的效果，很大程度上取决于主程序员的技术水平和管理才能。美国的软件产业中大多是主程序员制的工作方式。

② 民主制小组：组内成员之间可以平等地交换意见，工作目标的制定及做出决定都由全体成员参加。这种组织形式强调发挥小组每个成员的积极性，要求每个成员充分发挥主动精神和协作精神。有人认为这种组织形式适合于研制时间长、开发难度大的项目。日本在发展计算机事业中，组织软件开发大多采用这种形式的开发小组，取得了很好的效果。 ③ 层次式小组：这种结构比较适合项目本身就是层次结构状的课题。因为这样可以把项目按功能划分成若干个子项目，把子项目分配给基层小组，由基层小组完成。基层小组的领导与项目负责人直接联系。这种组织方式比较适合于大型软件项目的开发。

以上三种组织形式可以根据实际情况，组合起来灵活运用。例如，较大的软件项目也许是把主程序员小组组织成层次式结构；也许基层小组的领导又是一个民主制小组的成员。

第十章 软件质量保证

一、复习要求

1. 了解软件质量保证、质量保证活动与质量检验的概念。

2. 了解软件质量保证体系与质量保证的实施的概要。

3. 了解正式技术评审概要。包括评审会议、设计质量和程序质量的评审。

4. 了解软件配置管理的概念。包括配置项和基线概念、配置管理的主要工作。

5. 了解软件工程标准化的概念。包括软件工程标准化意义、软件工程标准的制定与推行、软件工程标准的层次、软件工程的国家标准。

6. 了解软件文档的概念。包括文档编制的要求、文档的作用、分类、文档的工作。

7. 了解软件过程与过程改进的概念。包括过程分类与过程模型、剪裁过程、过程模型建造技术、软件过程改进。

8. 了解软件过程能力评估的 CMM 模型，包括过程成熟度的概念、软件机构的能力成熟度模型、关键过程域、关键实践的概念。

9. 了解 ISO 9000 国际标准。包括质量管理、质量认证和质量审核的概念，ISO 9000 系列标准的特点、科学依据、主要内容，以及 ISO 9000-3 标准。

二、内容提要

1. 软件质量保证

(1) 质量保证的概念

什么是质量保证？它是为保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动。质量保证是面向消费者的活动，是为了使产品实现用户要求的功能，站在用户立场上来掌握产品质量的。这种观点也适用于软件的质量保证。 软件的质量保证就是向用户及社会提供满意的高质量的产品。进一步地，软件的质量保证活动也和一般的质量保证活动一样，是确保软件产品在软件生存期所有阶段的质量的活动。即为了确定、达到和维护需要的软件质量而进行的所有有计划、有系统的管理活动。它包括的主要功能如：

§ 制定和展开质量方针；

§ 制定质量保证方针和质量保证标准；

§ 建立和管理质量保证体系；

§ 明确各阶段的质量保证业务；

§ 坚持各阶段的质量评审；

§ 确保设计质量；

§ 提出与分析重要的质量问题；

§ 总结实现阶段的质量保证活动；

§ 整理面向用户的文档、说明书等；

§ 鉴定产品质量，鉴定质量保证体系；

§ 收集、分析和整理质量信息。

(2) 软件质量保证(SQA)活动

软件质量保证由各项任务构成，这些任务的参与者有两种人：软件开发人员和质量保证人员。前者负责技术工作，后者负责质量保证的计划、监督、记录、分析及报告工作。

软件开发人员通过采用可靠的技术方法和措施，进行正式的技术评审，执行计划周密的软件测试来保证软件产品的质量。软件质量保证人员则辅助软件开发组得到高质量的最终产品。1993 年美国 SEI 推荐了一组有关质量保证的计划、监督、记录、分析及报告的 SQA 活动。这些活动将由一个独立的 SQA 小组执行(或协助)：

① 为项目制定 SQA 计划。该计划在制定项目计划时制定，由相关部门审定。它规定了软件开发小组和质量保证小组需要执行的质量保证活动，其要点包括：需要进行哪些评价？需要进行哪些审计和评审？项目采用的标准；错误报告的要求和跟踪过程；SQA 小组应产生哪些文档？为软件项目组提供的反馈数量等。

② 参与开发该软件项目的软件过程描述。软件开发小组为将要开展的工作选择软件过程，SQA 小组则要评审过程说明，以保证该过程与组织政策、内部的软件标准、外界所制定的标准(如 ISO 9001）以及软件项目计划的其它部分相符。

③ 评审各项软件工程活动，核实其是否符合已定义的软件过程。SQA 小组识别、记录和跟踪所有偏离过程的偏差，核实其是否已经改正。

④ 审计指定的软件工作产品，核实其是否符合已定义的软件过程中的相应部分。SQA 小组对选出的产品进行评审，识别、记录和跟踪出现的偏差，核实其是否已经改正，定期向项目负责人报告工作结果。

⑤ 确保软件工作及工作产品中的偏差已被记录在案，并根据预定规程进行处理。偏差可能出现在项目计划、过程描述、采用的标准或技术工作产品中。

⑥ 记录所有不符合部分，并向上级管理部门报告。跟踪不符合的部分直到问题得到解决。

除了进行上述活动外，SQA 小组还需要协调变更的控制与管理，并帮助收集和分析软件度量的信息。

(3) 质量保证与检验

① 检验在质量保证中的作用

软件质量必须在设计和实现过程中加以保证。如果过程管理不力，软件开发环境或软件工具不够好，或者由于各种失误导致产生软件差错，其结果就会产生软件失效。为了确保每个开发过程的质量，防止把软件差错传递到下一个过程，必须进行质量检验。

质量保证是面向消费者的，从质量保证的角度来讨论检查，下面几点应当明确：

§ 用户要求的是产品所具有的功能，这是“真质量”。靠质量检验，一般检查的是“真质量”的质量特性。

§ 能靠质量检验的质量特性，即使全数检验，也只是代表产品的部分质量特性。

§ 必须在各开发阶段对影响产品质量的因素进行切实的管理，认真检查实施落实情况。只有这样才能使产品达到用户要求，这比单靠检验来保证质量要有效、经济。

§ 当开发阶段出现异常时，要从质量特性方面进行检验，看是否会给后续阶段带来影响，并对判断其好坏程度。从质量保证角度来看，此项工作极其重要。

§ 虽然各开发阶段进展稳定，但由于工具支持不足等，软件产品不能满足用户要求的质量。这时可通过检验对该产品做出评价，判断是否能向用户提供该产品。

§ 尽管各开发阶段进展稳定，但也要以一定的标准检验产品，使其交付使用后保持稳定的质量水平。同时还要根据产品的质量特性，检查各个过程的管理状态。

因此，检验的目的有二个。其一是切实搞好开发阶段的管理，检查各开发阶段的质量保证活动开展得如何；其二是预先防止软件差错给用户造成损失。

② 各个开发阶段中的检验

为了切实做好质量保证，要在软件开发工程的各个阶段实施检验。检验的类型有：

§ 供货检验：这是指对委托外单位承担开发作业，而后买进或转让的构成软件产品的部件、规格说明、半成品或产品的检查。由于委托单位、委托时间等情况差别很大，往往与质量相关的信息不完全，要想只靠供货时检查，质量很难保证。因此要调查接受委托单位的开发能力，并且要充分交流情况。

§ 中间检验／阶段评审：在各阶段的中途或向下一阶段移交时进行的检查叫做中间检验或阶段评审。阶段评审的目的是为了判断是否可进入下一阶段进行后续开发工作，避免将差错传播到后续工作中，给后续工作带来不良影响，造成损失。

§ 验收检验：确认产品是否已达到可以进行“产品检验”的质量要求。

§ 产品检验：这是软件产品交付使用前进行的检查。其目的是判定向用户提供的软件，作为产品，是否达到了令人满意的程度。

检验的实施有两种形式：实际运行检验(即白盒测试和黑盒测试)和鉴定。可在各开发阶段中结合起来使用。各开发阶段及阶段中的检验如图 10.1 所示

开发阶段		检查项目
需求分析	需求分析 ↓ 功能设计 ↓ 实施计划	① 开发目的. ② 目标值. ③ 开发量(程序、文档). ④ 所需资源. ⑤ 各阶段的产品、作业内容. ⑥ 开发体制. 的合理性
设计	结构设计 ↓ 数据设计 ↓ 过程设计	① 产品的量(计划量、实际量). ② 评审量. ③ 差错数. ④ 检出差错的内容和倾向. ⑤ 评审方法、覆盖性. ⑥ 出错原因、处理情况及对该阶段的影响. ⑦ 评审结束、阶段结束的判断标准
实现	程序编制 ↓ 单元测试 ↓ 组装测试 ↓ 确认测试	① 产品的量(计划量、实际量)，目标值完成情况. ② 评审量. ③ 检出的差错数. ④ 计算机使用时间. ⑤ 出错原因、处理情况及对该阶段的影响. ⑥ 检出的差错内容及倾向. ⑦ 评审方法、覆盖性. ⑧ 测试环境. ⑨ 测试项目设定种类、测试用例设计方法. ⑩ 评审结束、阶段结束的判断标准.
验收	检查，评价	① 说明书检查——检查与被检查程序有关的用户文档等. ② 程序检查——为了评价和保证程序质量，通过各种测试测试成品进行检查.
运行维护	运行，维护	① 掌握用户使用产品的质量情况，并反馈到开发部门.

图 10.1 开发阶段与相应的检验项目

2. 软件质量保证体系与质量保证的实施

(1) 软件质量保证体系

软件的质量保证活动，是涉及各个部门的部门间的活动。例如，如果在用户处发现了软件故障，产品服务部门就应听取用户的意见，再由检查部门调查该产品的检验结果，进而还要调查软件实现过程的状况，并根据情况检查设计是否有误，不当之处加以改进，防止再次发生问题。为了顺利开展以上活动，事先明确部门间的质量保证业务，确立部门间的联合与协作的机构十分重要，这个机构就是质量保证体系。图 10.2 和 10.3 是软件质量保证体系的图例。

在质量保证体系图上，用户、领导、各部门横向安排，而纵向则顺序列出软件质量保证活动的各项工作。制定质量体系保证图应注意以下一些问题：

§ 必须明确反馈途径。

§ 必须在体系图的纵向(纵坐标方向)顺序写明开发阶段，在横向(横坐标方向)写明组织机构，明确各部门的职责。

§ 必须确定保证系统运行的方法、工具、有关文档资料，以及系统管理的规程和标准。

§ 必须明确决定是否可向下一阶段进展的评价项目和评价准则。

§ 必须不断地总结系统管理的经验教训，能够修改系统。

仅靠质量保证体系图很难明确具体工作，因此必须制定质量保证计划，在这个计划中确定质量目标，确定在每个阶段为达到总目标所应达到的要求，对进度做出安排，确定所需的人力、资源和成本等等。

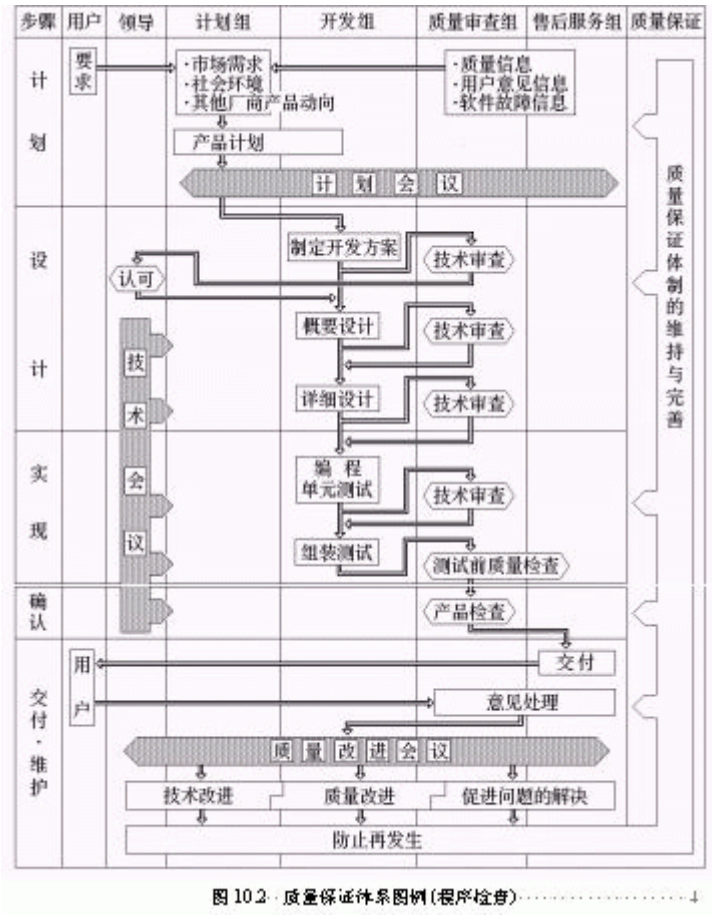


图 10.2 质量保证体系图例(程序检查)

在这个质量保证计划中包括的软件质量保证规程和技术准则应当

§ 指示在何时、何处进行文档检查和程序检查；

§ 指示应当采集哪些数据，以及如何进行分析处理，例如，在每次评审和测试中发现的错误如何修正；

§ 描述希望得到的质量度量；

§ 规定在项目的哪个阶段进行评审及如何评审；

§ 规定在项目的哪个阶段应当产生哪些报告和计划；

§ 规定产品各方面测试应达到的水平。

在计划中，在说明各种软件人员的职责时，要规定为了达到质量目标他们必须进行哪些活动。其次，要根据这个质量保证体系图，建立在各阶段中执行质量评价的质量评价和质量检查系统及有效运用质量信息的质量信息系统，并使其运行。

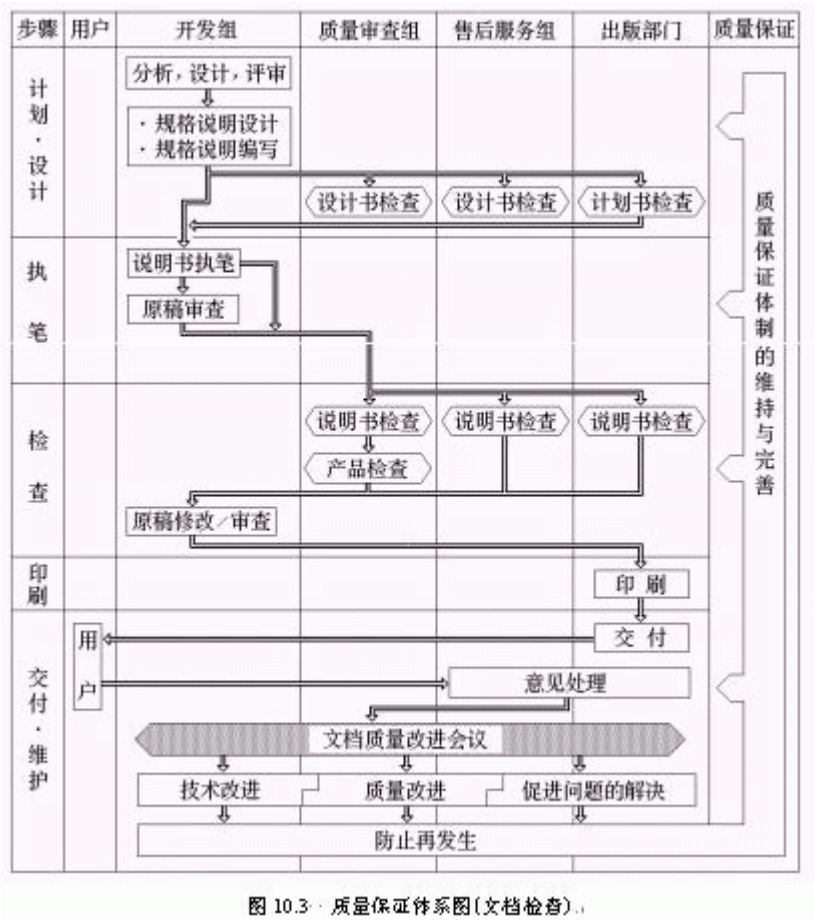


图 10.3 质量保证体系图(文档检查)

(2) 质量保证的实施

软件质量保证的实施需要从纵向和横向两个方面展开。一方面要求所有与软件生存期有关的人员都要参加，另一方面要求对产品形成的全过程进行质量管理，这要求整个软件部门齐心协力，不断完善软件的开发环境。此外还需要与用户共同合作。

① 质量目标与度量

为了开发高质量的软件，从计划阶段开始，不但需要明确软件的功能，还要明确软件应达到什么样的质量标准，即制定软件的质量目标。为了达到这个目标，在开发过程中的各个阶段进行检查和评价。在做质量评价时，需要有对质量进行度量的准则和方法，但更重要的是，需要有在软件生存期中如何使用这些准则和方法的质量保证步骤，以及提高该项作业效率的工具。

软件质量度量和保证的条件通常有以下几项：

§ 适应性：必须制定能适应各种用户要求、软件类型和规模的质量标准，并能够度量。

§ 易学性：不需要特殊技术，软件技术人员人人都容易掌握。

§ 可靠性：对同一软件的评价，尽管评价的人或场合可能不同，但评价结果必须一致。

§ 针对性：不是在检查时才改进质量，而必须从设计阶段起就确立质量目标，在各个阶段实施落实。

§ 客观性：从各种不同角度加以评价，并将评价结果定量地表示，使得人人都能理解。

§ 经济性：考虑如何才能把质量度量和保证所需要的费用控制在适当的范围内。

② 软件质量度量与保证的实施

图 10.4 给出软件质量度量和保证系统在质量保证活动中的五个实施步骤：

§ Target：以用户要求和开发方针为依据，对质量需求准则、质量设计准则的各质量特性设定质量目标。对各准则的重要程度可以设“特别重要”、“重要”、“一般”三级。

§ Plan：设定适合于被开发软件的评测检查项目，与此同时还要研讨实现质量目标的方法或手段。

§ Do：在开发标准和质量评价准则的指导下，制作高质量的规格说明书和程序。在接受质量检查之前要先做自我检查。

§ Check：以 Plan 阶段设定的质量评价准则进行评价。算出得分，用质量图的形式表示出来，参看图 10.5。比较评价结果的质量得分和质量目标，看其是否合格。

§ Action：对评价发现的问题进行改进活动，如果实现并达到了质量目标就转入下一个工程阶段。这样重复“Plan”到“Action”的过程，直到整个开发项目完成。

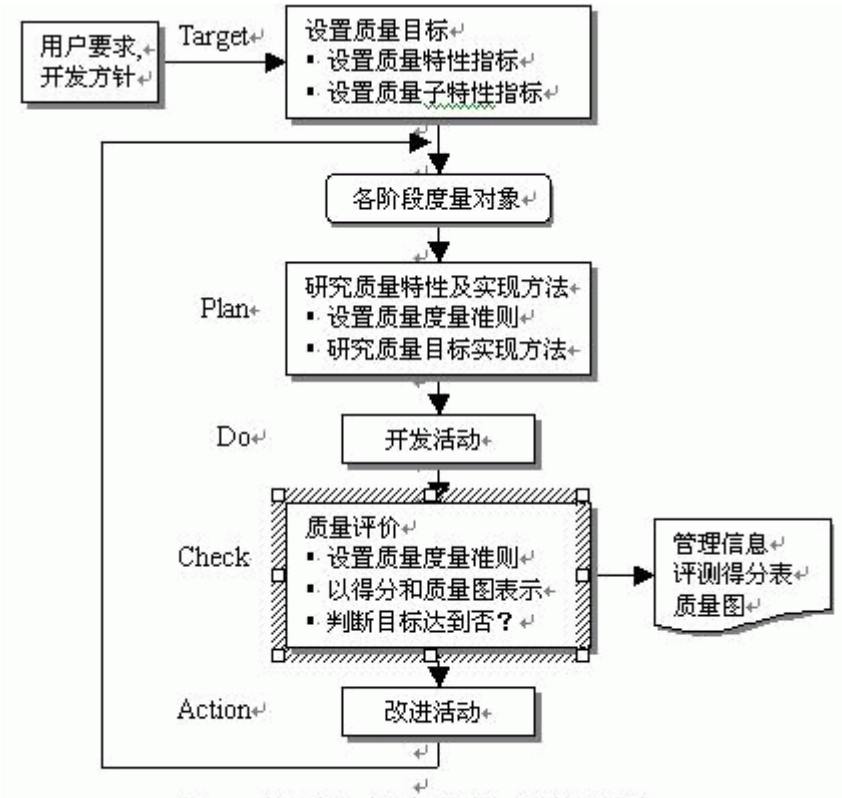
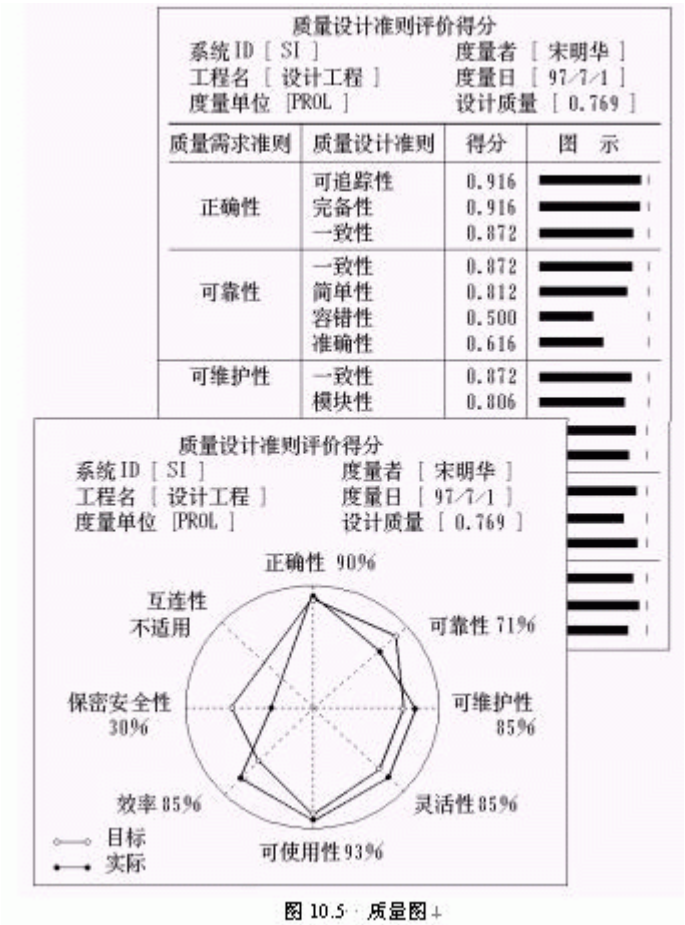


图 10.4 软件质量度量与保证体系的管理周期



质量设计准则评价得分

系统 ID [SI] 度量者 [宋明华]
工程名 [设计工程] 度量日 [97/7/1]
度量单位 [PROL] 设计质量 [0.769]

质量需求准则	质量设计准则	得分	图 示
正确性	可追踪性	0.916	
	完备性	0.916	
	一致性	0.872	
可靠性	一致性	0.872	
	简单性	0.812	
	容错性	0.500	
	准确性	0.616	
可维护性	一致性	0.872	
	模块性	0.806	

图 10.5 质量图

整理人：刘海云
整理时间：2003.05

3. 正式技术评审（Formal Technical Review，FTR）

人的认识不可能100%符合客观实际，因此在软件生存期每个阶段的工作中都可能引入人为的错误。在某一阶段中出现的错误，如果得不到及时纠正，就会传播到开发的后续阶段中去，并在后续阶段中引出更多的错误。实践证明，提交给测试阶段的程序中包含的错误越多，经过同样时间的测试后，程序中仍然潜伏的错误也越多。所以必须在开发时期的每个阶段，特别是设计阶段结束时都要进行严格的技术评审，尽量不让错误传播到下一个阶段。

软件技术评审是软件开发人员实施的一种质量保证活动，FTR的目标是：

§ 针对任一种软件范型，发现软件在功能、逻辑和实现上的错误。

§ 验证经过评审的软件确实满足需求。

§ 保证软件是按照已确定的标准表述的。

§ 使得软件能按一致的方式开发。

§ 使软件项目跟容易管理。

此外，FTR还起到了提高项目连续性和训练软件工程人员的作用。FTR 包括了“走查”、“检查”、“循环评审”和其它的软件评审技术。每次FTR都以会议形式进行，只有在很好地计划、控制和参与的情况下，FTR才有可能获得成功。

(1) 评审会议

每次评审会议都需遵守以下规定：

§ 每次会议的参加人数3 ~ 5人。

§ 会前应做好准备，但每个人的工作量不应超过2小时。

§ 每次会议的时间不应超过2小时。

按照上述规定，显然FTR关注的应是整个软件的某一特定(且较小)的部分。例如，不是对整个设计评审，而是逐个模块走查，或走查模块的一部分。通过缩小关注的范围，更容易发现错误。

FTR的关注点集中于某个工作产品，即软件的某一部分(如部分需求规格说明、一个模块的详细设计、一个模块的源程序清单）。评审会议由评审负责人主持，所有评审人员和开发人员参加。FTR首先讨论日程安排，然后让开发人员“遍历”其工作产品，做简单介绍。评审人员根据他们事先的准备提出问题。当问题被确认或错误被发现，记录员要将其一一记录下来。评审结束时，所有FTR的参加者必须作出决定：

§ 接受该工作产品，不再做进一步的修改。

§ 由于该工作产品错误严重，拒绝接受(错误改正后必须再次进行评审)。

§ 暂时接受该工作产品(发现必须改正的微小错误，但不必再次进行评审)。

当决定之后，FTR 的所有参加者都必须签名，以表明他参加了会议，并同意评审组的决定。

(2) 评审内容

通常，把“质量”定义为“用户的满意程度”。为使得用户满意，有两个必要条件：

§ 设计的规格说明要符合用户的要求；

§ 程序要按照设计规格说明所规定的情况正确执行。

我们把上述条件(1)称为“设计质量”，把条件(2)称为“程序质量”。

与上述质量的观点相对应，软件的规格说明可以分为外部规格说明和内部规格说明。外部规格说明是从用户角度来看的规格，包括硬件 / 软件系统设计(在分析阶段进行)、功能设计(在需求分析阶段与概要设计阶段进行)，而内部规格说明是为了实现外部规格的更详细的规格，即程序模块结构设计与模块处理加工设计(在概要设计与详细设计阶段进行)。因此，内部规格说明是从开发者角度来看的规格说明。将上述两个概念联系起来，则可以说设计质量是由外部规格说明决定的，程序质量是由内部规格说明决定的。

下面讨论针对外部规格说明进行的设计评审。评审对象是在需求分析阶段产生的软件需求规格说明、数据要求规格说明，在软件概要设计阶段产生的软件概要设计说明书等。通常，需要从 12 个方面进行评审。

§ 评价软件的规格说明是否合乎用户的要求：

§ 评审可靠性措施是否能避免引起系统失效的原因发生，而一旦发生后能否及时采取代替手段或恢复手段。

§ 评审保密措施是否能实现。

§ 评审操作特性实施情况。可从 4 个方面检查：操作命令和操作信息的恰当性；输入数据和输入控制语句的恰当性；输出数据的恰当性和应答时间的恰当性。

§ 评审性能实现情况。一般来说，因性能设计是需要考虑多方面因素的复杂工作，因此，应明确规定性能的目标值。性能目标设定条件的恰当性；明确性能的评价方法。

§ 评审软件是否具有可修改性。需要考察系统是否具备以下功能：检测故障的功能，获取分析数据的功能；区分问题根源的方法；故障修正的方法。

§ 评审软件是否有可扩充性。

§ 评审软件是否具有兼容性。

§ 评审软件是否具有可移植性。

§ 评审软件是否具有可测试性：为保证软件在修改或扩充后的正确性，不仅要测试被修改或被扩充的部分是否能按规格执行，而且还应对该软件原有的功能经修改或扩充后是否能按以前的规格正确运行进行测试。

§ 评审软件是否具有可复用性。

§ 评审软件是否具有互连性：要求软件与其它软件应有共同接口，与其它软件之间的接口部分应是模块化的。

程序质量评审是着眼于软件本身的结构、与运行环境的接口、变更带来的影响而进行的评审活动。通常它是从开发者的角度进行评审，直接与开发技术有关。

§ 软件的结构。需要检查的项目有：数据结构、功能结构、数据结构和功能结构之间的对应关系。

§ 功能的通用性。

§ 模块层次。包括模块的层次结构，与功能层次的对应关系。

§ 模块结构。检查的项目有：控制流结构、数据流结构、模块结构与功能结构之间的对应关系，包括功能结构与控制流结构的对应关系；功能结构与数据流结构的对应关系。以及每个模块的定义。

§ 处理过程的结构。对它的检查项目有：要求模块的功能结构与实现这些功能的处理过程的结构应明确对应；要求控制流应是结构化的；数据的结构与控制流之间的对应关系应是明确的，并且可依这种对应关系来明确数据流程的关系。

§ 与运行环境的接口。主要检查项目有：与其它软件的接口；与硬件的接口；与用户的接口；变更的影响范围问题。

4. 软件配置管理

在软件建立时变更是不可避免的，而变更更加剧了项目中软件人员之间的混乱。之所以产生混乱，是因为在进行变更前没有仔细分析，或没有进行变更控制。Babich 曾经这样说过：“协调软件开发使得混乱减到最小的技术叫做配置管理。配置管理是一种标识、组织和控制修改的技术，目的是使错误达到最小并最有效地提高生产率”。

软件配置管理，简称 SCM，是一种“保护伞”活动，它应用于整个软件生存期。因为变更在任何时刻都可能发生，因此软件配置管理活动的目标就是为了标识变更，控制变更，确保变更正确地实现，向其他有关的人报告变更。 软件维护和软件配置管理之间的区别是：维护是一组软件工程活动，它们发生于软件已交付给用户并已投入运行之后；软件配置管理是一组追踪和控制活动，它们开始于软件开发项目开始之时，结束于软件被淘汰之时。

(1) 软件配置项(Software Configuration Item，简称 SCI)

软件工程过程的输出有三种信息：计算机程序(源程序及目标程序)，描述计算机程序的文档(包括技术文档和用户文档)，数据结构。在软件工程过程中产生的所有的信息项(文档、报告、程序、表格、数据)就构成了软件配置。软件配置是软件的具体形态在某一时刻的瞬时影像。这样的具体形态取两种形式：

§ 不可直接执行的材料：如书写的文档、程序清单、测试数据、测试结果等。

§ 可直接执行的材料：如目标代码、数据库信息等。它们可由计算机处理，存于某种存储介质上。

软件配置管理的对象就是软件配置项，它们是软件工程过程中产生的信息项。按照 ISO 9000-3 的说明，软件配置项可以是：

§ 与合同、过程、计划和产品有关的文档和数据；

§ 源代码、目标代码和可执行代码；

§ 相关产品，包括软件工具、库内的可复用软件、外购软件及用户提供的软件。

软件工具包括编辑程序、编译程序、其它 CASE 工具的特定版本，都要做为软件配置的一部分加以“冻结”。如果编译程序的版本不同，可能产生的结果也不同。

随着软件工程过程的进展，软件配置项数目快速增加。如果每个配置项只是简单地产生其它软件配置项，造成的混乱可能微乎其微。然而在变更时会引入其它影响因素，使得情况变得更加复杂。这时配置管理的作用就会充分显示出来。

不仅如此，所有以上提到的软件配置项在不同时期，出于不同的要求进行了各种组合，如针对不同的硬件环境和软件环境的各种组合，这就是软件配置的概念。在实现软件配置管理时，把软件配置项组织成配置对象，在项目数据库中用一个单一的名字来组织它们。一个配置对象有一个名字和一组属性，并通过某些联系“连接”到其它对象，如图 10.6 所示。图中分别对配置对象进行了定义，每个对象与其它对象的联系用箭头表示。箭头标明了构成关系。如“数据模型”和“模块 N”是“设计规格说明”的一部分。双向箭头则表明一种相互关系。如果对“源代码”对象作了一个变更，软件人员可以根据这种相互关系确定其它哪些对象可能受到影响。

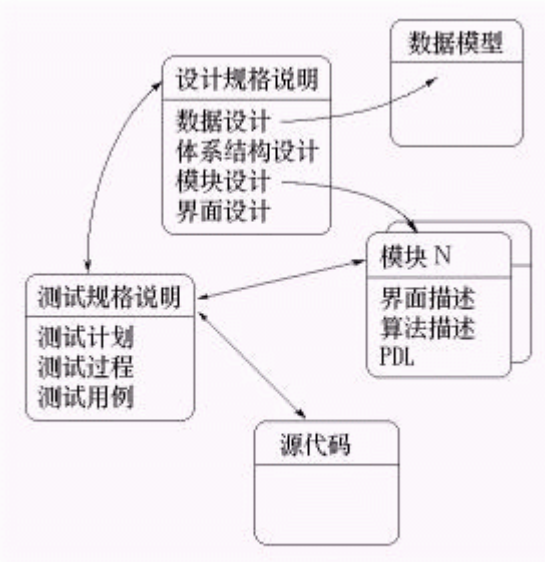


图 10.6 配置对象

(2) 基线 (Baseline)

基线是软件生存期中各开发阶段末尾的特定点，又称里程碑。由正式的技术评审而得到的软件配置项协议和软件配置的正式文本才能成为基线。它的作用是把各阶段工作的划分更加明确化，使本来连续的工作在这些点上断开，以便于检验和肯定阶段成果。例如，明确规定不允许跨越里程碑修改另一阶段的文档。如图 10.7 所示，是软件开发各阶段的基线。

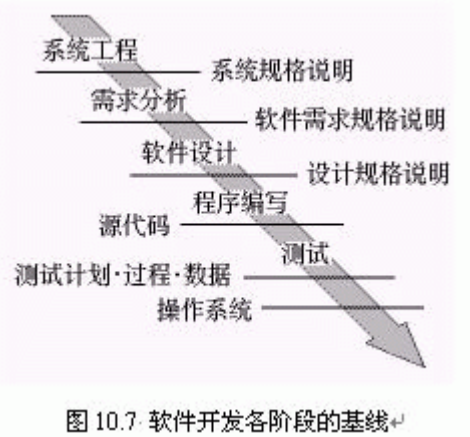


图 10.7 软件开发各阶段的基线

一旦一个软件配置项成为基线，就把它存放到项目数据库(亦称项目信息库或软件仓库)中。当一位软件组织成员想要对基线配置项进行修改时，就把它从项目数据库中复制到该工程师的专用工作空间中，如图 10.8 所示。图中把一个标号为 B 的配置项从项目数据库复制到工程师的专用工作空间中。这个活动记录在一个记事文件中。工程师可以在 B’(B 的副本)上完成要求的变更，然后用 B’来更新 B。有些系统中把这个基线配置项锁定，在变更完成、评审和批准之前，不许对它做任何操作。

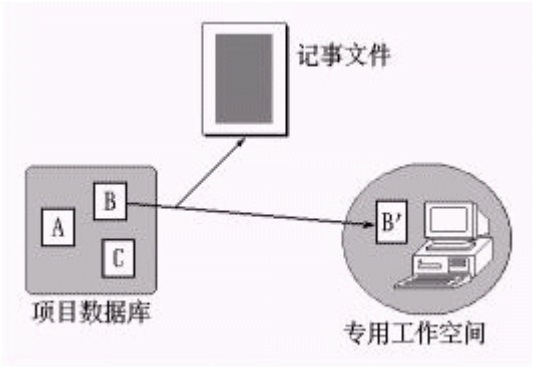


图 10.8 基线 SCI 和项目数据库

(3) 软件配置管理的过程

软件配置管理除了担负控制变更的责任之外，它还要担负标识单个的软件配置项和软件各种版本、审查软件配置以保证开发得以正常进行，以及报告所有加在配置上的变更等任务。

有关软件配置管理，需要考虑这样一些问题：

§ 采用什么方式来标识和管理许多已存在程序(和它们的文档)的各种版本？

§ 在软件交付用户之前和之后如何控制变更？

§ 谁有权批准和对变更安排优先级？

§ 如何保证变更得以正确地实施？

§ 利用什么办法来估计变更可能引起的其它问题？

这些问题归结到软件配置管理的 5 个任务，即配置标识、版本管理、变更控制、配置审核和配置报告。

(4) 配置标识

软件配置实际上是一个动态的概念。一方面随着软件生存期的向前推进，软件配置项的数量在不断增多。另一方面又随时会有新的变更出现，形成新的版本。因此，整个软件生存期的软件配置就象一部不断演变的电影，而某一时刻的配置就是这部电影的一个片段。 为了方便对软件配置的各个片段，即软件配置项进行控制和管理，不致造成混乱，首先应给它们命名，这就是配置标识的任务。

我们可以用面向对象的方法进行标识。通常需标识两种类型的对象：基本对象和复合对象。基本对象是由软件人员在分析、设计、编码和测试时所建立的“文本单元”。例如，基本对象可能是需求规格说明中的一节，一个源程序清单、一组用来测试一个等价类的测试用例。复合对象则是基本对象或其它复合对象的一个

集合。如图 10.6 所示，“设计规格说明”是一个复合对象，它是一些基本对象，如“数据模型”、“模块 N”的集合。

每个对象可用一组信息来唯一地标识它，这组信息包括：

（名字、描述、一组“资源”、“实现”）

对象的名字是一个字符串，它明确地标识对象。对象描述是一个表项，它包括：对象所表示的配置项类型(如文档、程序、数据)、项目标识、变更和 / 或版本信息。资源是“由对象所提供的、处理的、引用的或其它所需要的一些实体”。例如，数据类型，特定函数，甚至变量名都可以看做是对象资源。而“实现”对于一个基本对象来说，是指向“文本单元”的指针，而对于复合对象来说，则为 null。

配置对象的标识还需考虑在命名对象间存在的联系。对象可以是一个复合对象的组成部分，用联系<part of>进行标识。这个联系定义了对象的层次。在对象层次中，对象之间的联系不仅存在于层次树的路径中，而且可跨越对象层次的分支相互关联。例如，数据模型与数据流图是相互关联的，而且它又与一个特定等价类的测试用例组相互关联。

(5) 版本控制

整个软件工程过程中所涉及的软件对象都必须加以标识。在对象成为基线以前可能要做多次变更，在成为基线之后也可能需要频繁地变更。这样对于每一配置对象都可以建立一个演变图，这个演变图记叙了对象的变更历史。以图 10.9 为例，配置对象 1.0 经过修改成为对象 1.1，又经历了小的修改和变更，产生了版本 1.1.1 和 1.1.2。紧接对版本 1.1 做了一次更新，产生对象 1.2，又持续演变生成了 1.3 和 1.4 版本。同时对对象 1.2 做了一次较大的修改，引出一条新的演变路径：版本 2.0。

已经开发出来许多工具来辅助标识工作。在某些工具中，当前保持的只是最后版本的完全副本。为了得到较早时期(文档或程序)的版本，可以从最后版本中“提取”出(由工具编目的)变更，使得当前配置直接可用，并使得其它版本也可用。

在图 10.9 所示的演变图中，各结点都是聚合对象。软件的每一版本都是软件配置项(源代码、文档、数据)的一个集合，且各个版本都可能由不同的变种组成。例如，有一个简单的程序版本：它由 1、2、3、4 和 5 等部件组成，其中部件 4 在软件使用彩色显示器时使用，部件 5 在软件使用单色显示器时使用。因此，可以定义版本的两个变种：

① 部件 1、2、3、4； ② 部件 1、2、3、5。

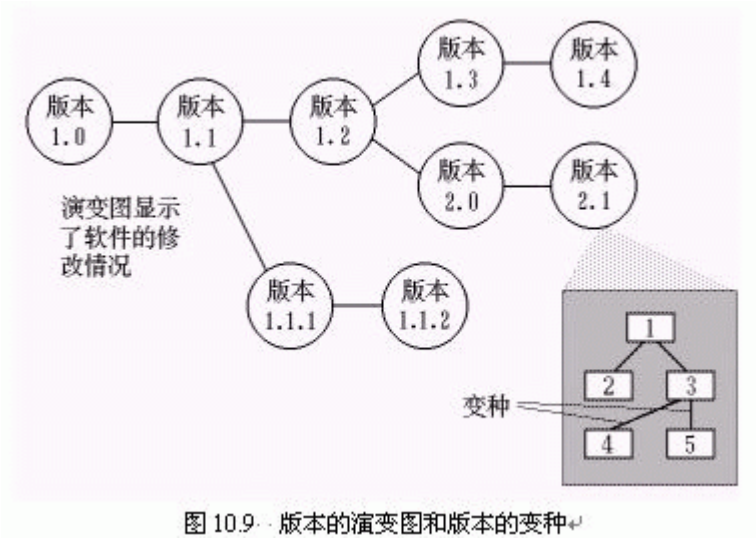


图 10.9 版本的演变图和版本的变种

版本控制的主要功能有：

① 集中管理档案，安全授权机制：版本管理的操作是将开发组的档案集中存放在服务器上，经系统管理员授权给各个用户。用户通过登入(check in)和检出(check out)的方式访问服务器上的文件，未经授权的用户则无法访问服务器上的文件。如图 10.10 所示。

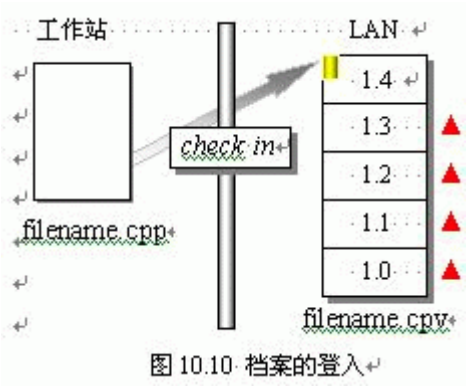


图 10.10 档案的登入

② 软件版本升级管理：每次登入时，在服务器上都会生成新的版本，软件版本的管理采取增量存储的方式。任何版本都可以随时检出编辑，同一应用的不同版本可以像树枝一样向上增长。如图 10.11 所示。



图 10.12 变更控制过程

③ 加锁功能：为了在文件更新时保护文件，避免不同的用户更改同一文件时发生冲突。某一文件一旦被登入，锁即被解除，该文件可被其它用户使用。在更新一个文件之前锁定它，避免变更没有锁定的项目源文件。

④ 提供不同版本源程序的比较：在文件登入和检出时，需要注意检入和检出的使用。

§ 当某个时刻需要修改某个小缺陷或特征，应只检出完成工作必需的最少文件；

§ 当需要对文件变更时，应登入它并加锁。这样可保留对每个变更的记录；

§ 应避免长时间地锁定文件。如果需要长时间工作于某个文件，最好能创建一个分支，并在分支上做工作。这样别人可以地做缺省的“小”修改以更改较小的缺陷。稍后可通过合并，把所有操作结果集成在一起。

§ 如果需要做较大的变更，可有两种选择：

ⅰ）将需要的所有文件检出并加锁，然后正常处理；

ⅱ）为需要修改的版本创建分支，把变更与主干“脱离”，然后把结果合并回去。

(6) 变更控制

软件生存期内全部的软件配置是软件产品的真正代表，必须使其保持精确。软件工程过程中某一阶段的变更，均要引起软件配置的变更，这种变更必须严格加以控制和管理，保持修改信息，并把精确、清晰的信息传递到软件工程过程的下一步骤。

变更控制包括建立控制点和建立报告与审查制度。

对于一个大型的软件来说，不加控制的变更很快就会引起混乱。因此变更控制是一项最重要的软件配置任务。图 10.12 给出了变更控制的过程。

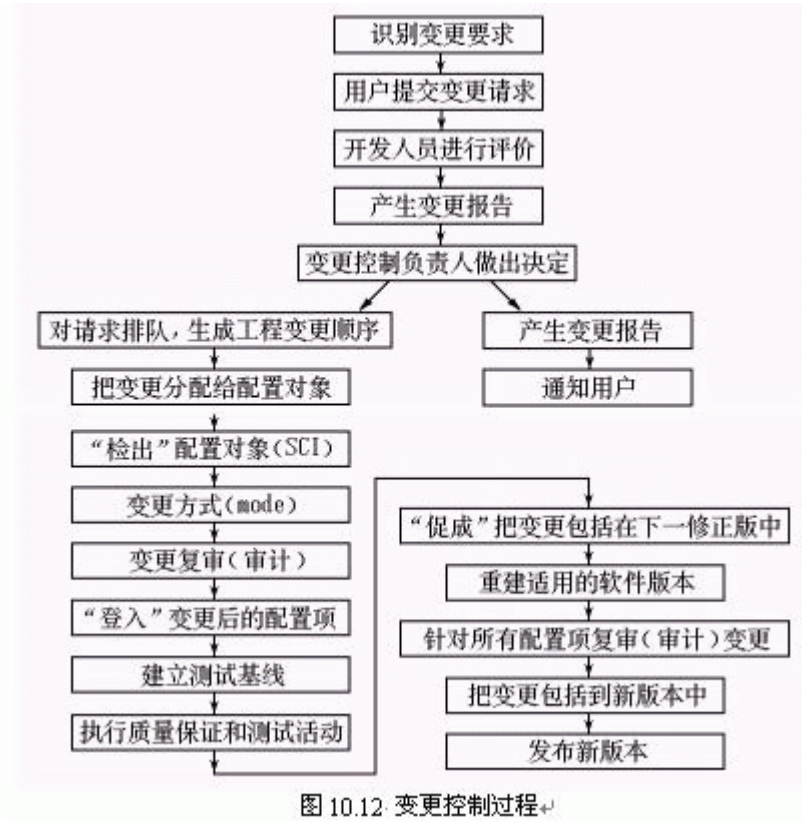


图 10.12 变更控制过程

在此过程中，首先用户提交书面的变更请求，详细申明变更的理由、变更方案、变更的影响范围等。然后由变更控制机构确定控制变更的机制、评价其技术价值、潜在的副作用、对其它配置对象和系统功能的综合影响以及项目的开销、并把评价的结果以变更报告的形式提交给变更控制负责人(最终决定变更状态和优先权的某个人或小组)。对每个批准了的变更产生一个工程变更顺序(ECO)，描述进行的变更、必须考虑的约束、评审和审计的准则等。要做变更的对象从项目数据库中检出，对其做出变更，并实施适当的质量保证活动。然后再把对象登入到数据库中并使用适当的版本控制机制建立软件的下一版本。

“检出”和“登入”处理实现了两个重要的变更控制要素，即存取控制和同步控

制。存取控制管理人们存取或修改一个特定软件配置对象的权限；同步控制可用来确保由不同的人所执行的并发变更不会产生混乱。

存取和同步控制流如图 10.13 所示。根据经批准的变更请求和 ECO，软件人员从项目数据库中检出要变更的配置对象。同步控制功能则封锁了项目数据库中的这个对象，使得当前检出的版本在没有被置换前不能再更新它。当然，对这个对象还可以检出另外的副本，但对其也不能更新。人们在对这种成为基线的对象做了变更，并经过适当的软件质量保证和测试之后，把修改版本登入项目数据库，再解除封锁。

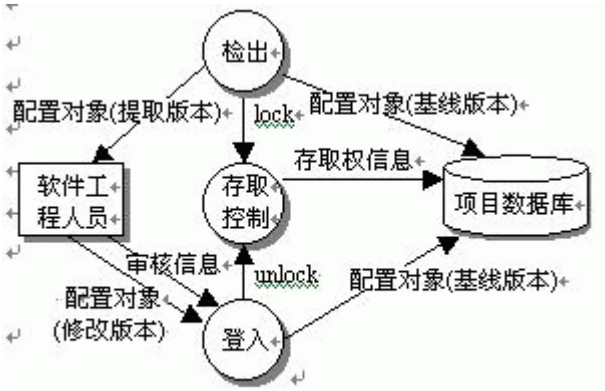


图 10.13 存取和同步控制

软件的变更通常有两类不同的情况：

§ 为改正小错误需要的变更。通常不需要从管理角度对这类变更进行审查和批准。但如果发现错误的阶段在造成错误的阶段的后面，例如在实现阶段发现了设计错误，则必须遵照标准的变更控制过程，把这个变更正式记入文档，并修改所有受这个变更影响的文档。

§ 为了增加或者删掉某些功能、或者为了改变完成某个功能的方法而需要的变更。这类变更必须经过某种正式的变更评价过程，以估计变更需要的成本和它对软件系统其它部分的影响。

应该把所做的变更正式记入文档，并相应地修改所有有关的文档。这种变更报告和审查制度，对变更控制来说起了一个安全保证作用。需要注意的是，必须对每一项变更进行评价并对所有的变更进行跟踪和复审。

(7) 配置状态报告

为了清楚、及时地记载软件配置的变化，不致于到后期造成贻误，需要对开发的过程做出系统的记录，以反映开发活动的历史情况。这就是配置状态登录的任务。登录主要根据变更控制小组会议的记录，并产生配置状态报告。报告对于每一项变更，记录以下问题：发生了什么？为什么会发生？谁做的？什么时侯发生的？会有什么影响？

图 10.14 描述了配置状态报告的信息流。

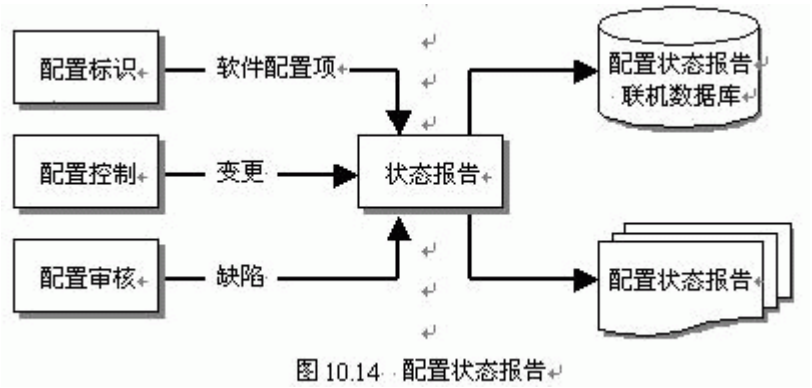


图 10.14 配置状态报告

每次新分配一个软件配置项或更新一个已有软件配置项的标识，或者一项变更申请被变更控制负责人批准，并给出了一个工程变更顺序时，在配置状态报告中就要增加一条变更记录条目。一旦进行了配置审核，其结果也应该写入报告之中。配置状态报告可以放在一个联机数据库中，以便软件开发人员或者软件维护人员可以对它进行查询或修改。此外在软件配置报告中新登录的变更应当及时通知给管理人员和软件人员。

配置状态报告对于大型软件开发项目的成功起着至关重要的作用。它提高了所有开发人员之间的通信能力，避免了可能出现的不一致和冲突。

(8) 配置审核

软件的完整性，是指开发后期的软件产品能够正确地反映用户所提出的对软件的要求。软件配置审核的目的就是要证实整个软件生存期中各项产品在技术上和管理上的完整性。同时，还要确保所有文档的内容变动不超出当初确定的软件要求范围。使得我们的软件配置具有良好的可跟踪性。这是软件变更控制人员掌握配置情况、进行审批的依据。

软件的变更控制机制通常只能跟踪到工程变更顺序产生为止，那么如何知道变更是否正确完成了呢？一般可以用以下两种方法去审查：正式技术评审和软件配置审核。

正式的技术评审着重检查已完成修改的软件配置对象的技术正确性，评审者评价软件配置项，决定它与其它软件配置项的一致性，是否有遗漏或可能引起的副作用。正式技术评审应对所有的变更进行，除了那些最无价值的变更之外。 软件配置审核作为正式技术评审的补充，评价在评审期间通常没有被考虑的软件配置项的特性。软件配置审核提出并解答以下问题：

§ 在工程变更顺序中规定的变更是否已经做了？每个附加修改是否已经纳入？

§ 正式技术评审是否已经评价了技术正确性？

§ 是否正确遵循了软件工程标准?

§ 在软件配置项中是否强调了变更? 是否说明了变更日期和变更者? 配置对象的属性是否反映了变更?

§ 是否遵循了标记变更、记录变更、报告变更的软件配置管理过程?

§ 所有相关的软件配置项是否都已正确地做了更新?

在某些情形下，这些审核问题是作为正式技术评审的一部分提出的。但是当软件配置管理成为一项正式活动时，软件配置审核就被分开，而由质量保证小组执行了。

5. 软件工程标准化

(1) 软件工程标准化的意义

在开发一个软件时，需要有许多层次、不同分工的人员相互配合；在开发项目的各个部分以及各开发阶段之间也都存在着许多联系和衔接问题。如何把这些错综复杂的关系协调好，需要有一系列统一的约束和规定。在软件开发项目取得阶段成果或最后完成时，还需要进行阶段评审和验收测试。投入运行的软件，其维护工作中遇到的问题又与开发工作有着密切的关系。软件的管理工作则渗透到软件生存期的每一个环节。所有这些都要求提供统一的行为规范和衡量准则，使得各种工作都能有章可循。

软件工程的标准化会给软件工作带来许多好处，比如：

§ 可提高软件的可靠性、可维护性和可移植性；

§ 可提高软件的生产率；

§ 可提高软件人员的技术水平；

§ 可提高软件人员之间的通信效率，减少差错和误解；

§ 有利于软件管理；有利于降低软件产品的成本和运行维护成本；

§ 有利于缩短软件开发周期。

随着人们对计算机软件的认识逐渐深入。软件工作的范围从只是使用程序设计语言编写程序，扩展到整个软件生存期。诸如软件概念的形成、需求分析、设计、实现、测试、安装和检验。运行和维护，直到软件淘汰(为新的软件所取代)。同时还有许多技术管理工作(如过程管理、产品管理、资源管理)以及确认与验证工作(如评审和审核、产品分析、测试等)常常是跨越软件生存期各个阶段的专门工作。所有这些方面都应当逐步建立起标准或规范来。另一方面，软件工程标准的

类型也是多方面的。根据中国国家标准 GB / T 15538－1995《软件工程标准分类法》，软件工程标准的类型有：

§ 过程标准：如方法、技术、度量等。

§ 产品标准：如需求、设计、部件、描述、计划、报告等。

§ 专业标准：如职别、道德准则、认证、特许、课程等。

§ 记法标准：如术语、表示法、语言等。

(2) 软件工程标准的制定与推行

软件工程标准的制定与推行通常要经历一个环状的生命周期，如图 10.15 所示。最初，制定一项标准仅仅是初步设想，经发起后沿着环状生命期，顺时针进行要经历以下的步骤：

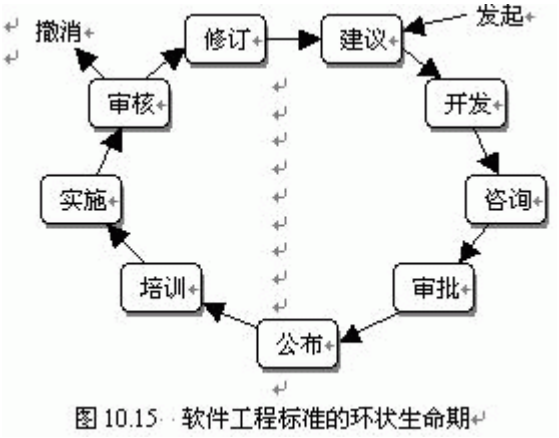


图 10.15 软件工程标准的环状生命期

图 10.15 软件工程标准的环状生命期

§ 建议：拟订初步的建议方案；

§ 开发：制定标准的具体内容；

审核

§ 咨询：征求并吸取有关人员的意见；

咨询

§ 审批：由管理部门决定能否推出；

§ 公布：公布发布，使标准生效；

§ 培训：为推行标准准备人员条件；

§ 实施：投入使用，需经历相当期限；

§ 审核：检验实施效果，决定修改还是撤消；

§ 修订：修改其中不适当的部分，形成标准的新版本，进入新的周期。

为使标准逐步成熟，可能在环状生命周期上循环若干圈，需要做大量的工作。

(3) 软件工程标准的层次

根据软件工程标准制定的机构和标准适用的范围有所不同，它可分为五个级别，即国际标准、国家标准、行业标准、企业(机构)标准及项目(课题)标准。以下分别对五级标准的标识符和标准制定(或批准)的机构做一简要说明：

① 国际标准

由国际联合机构制定和公布，提供各国参考的标准。如 ISO(International Standards Organization)——国际标准化组织。这一国际机构有着广泛的代表性和权威性，它所公布的标准也有较大的影响。1960 年代初，该机构建立了“计算机与信息处理技术委员会”，简称 ISO / TC97，专门负责与计算机有关的标准化工作。这一标准通常冠有 ISO 字样，如 ISO 8631－86 Information processing - program constructs and conventions for their representation《信息处理——程序构造及其表示法的约定》。该标准现已由中国收入国家标准。

② 国家标准

由政府或国家级的机构制定或批准，适用于全国范围的标准，如：

§ GB——中华人民共和国国家技术监督局是中国的最高标准化机构，它所公布实施的标准简称为“国标”。现已批准了若干个软件工程标准。

§ ANSI(American National Standards Institute)——美国国家标准协会。这是美国一些民间标准化组织的领导机构，具有一定的权威性。

§ FIPS(NBS) (Federal Information Processing Standards(National Bureau of Standards))——美国商务部国家标准局联邦信息处理标准。它所公布的标准均冠有 FIPS 字样。如 1987 年发表的 FIPS PUB 132－87 Guideline for validation and verification plan of computer software (软件确认与验证计划指南)。

§ BS(British Standard)——英国国家标准。

§ DIN(Deutsches Institut für Normung)——德国标准协会

§ JIS(Japanese Industrial Standard)——日本工业标准

③ 行业标准

由行业机构、学术团体或国防机构制定，并适用于某个业务领域的标准，如：

§ IEEE(Institute of Electrical and Electronics Engineers)——美国电气与电子工程师学会。近年该学会专门成立了软件标准分技术委员会(SESS)，积极开展了软件标准化活动，取得了显著成果，受到了软件界的关注。IEEE 通过的标准经常要报请 ANSI 审批，使之具有国家标准的性质。因此，日常看到 IEEE 公布的标准常冠有 ANSI 的字头。例如，ANSI / IEEE Str 828－1983《软件配置管理计划标准》。

§ GJB——中华人民共和国国家军用标准。这是由中国国防科学技术工业委员会批准，适合于国防部门和军队使用的标准。例如，1988 年实施的 GJB 437－88《军用软件开发规范》；GJB 438－88《军用软件文档编制规范》。

§3) DOD_STD(Department Of Defense_STanDards)——美国国防部标准，适用于美国国防部门。

§ MIL_S(MILitary_Standard)——美国军用标准，适用于美军内部。

此外，近年来中国许多经济部门(例如，原航空航天部、原国家机械工业委员会、对外经济贸易部、石油化学工业总公司等)都开展了软件标准化工作，制定和公布了一些适合于本部门工作需要的规范。这些规范大都参考了国际标准或国家标准，对各自行业所属企业的软件工程工作起了有力的推动作用。

④ 企业规范

一些大型企业或公司，由于软件工程工作的需要，制定适用于本部门的规范。例如，美国 IBM 公司通用产品部(General Products Division)1984 年制定的《程序设计开发指南》，仅供该公司内部使用。

⑤ 项目规范

由某一科研生产项目组织制定，且为该项任务专用的软件工程规范。例如，计算机集成制造系统(CIMS)的软件工程规范。

(4) 软件工程的国家标准

1983 年 5 月中国原国家标准总局和原电子工业部主持成立了“计算机与信息技术标准化技术委员会”，下设十三个分技术委员会。与软件相关的程序设计语言分委员会和软件工程技术分委员会。中国制定和推行标准化工作的总原则是向国际标准靠拢，对于能够在中国适用的标准一律按等同采用的方法，以促进国际交流。这里，等同采用是要使自己的标准与国际标准的技术内容完全相同，仅稍做

编辑性修改。 从 1983 年起到现在，中国已陆续制定和发布了 20 项国家标准。这些标准可分为 4 类：

① 基础标准； ② 开发标准； ③ 文档标准； ④ 管理标准。

在表 10.1 所示的表中分别列出了这些标准的名称及其标准号。除去国家标准以外，近年来中国还制定了一些国家军用标准。根据国务院、中央军委在 1984 年 1 月颁发的军用标准化管理办法的规定，国家军用标准是指对国防科学技术和军事技术装备发展有重大意义而必须在国防科研、生产、使用范围内统一的标准。凡已有的国家标准能满足国防系统和部队使用要求的，不再制定军用标准。

表 10.1 中国的软件工程标准

分类	标准名称	标准号	
基础标准	信息处理——数据流程图、程序流程图、系统流程图、程序网络图和系统资源图的文件编辑符号及约定	GB 1526—89	ISO 5807—1985
	软件工程术语	GB/T 11457—89	
	软件工程标准分类法	GB/T 15538—95	ANSI/IEEE 1002
	信息处理——程序构造及其表示法的约定	GB 13502—92	ISO 8631
	信息处理——单命中判定表的规范	GB/T 15535—95	ISO 5806
	信息处理系统——计算机系统配置图符号及其约定	GB/T 14085—93	ISO 8790
开发标准	软件开发规范	GB 8566—88	
	计算机软件单元测试	GB/T 15532—95	
	软件支持环境		
	信息处理——按记录组处理顺序文卷的程序流程		ISO 6593—1985
	软件维护指南	GB/T 14079—93	
文档标准	软件文档管理指南		
	计算机软件产品开发文件编制指南	GB 8567—88	
	计算机软件需求说明编制指南	GB 9385—88	ANSI/IEEE 829
	计算机软件测试文件编制规范	GB 9386—88	ANSI/IEEE 830
管理标准	计算机软件配置管理计划规范	GB/T 12505—90	IEEE 828
	信息技术 软件产品评价 质量特性及其使用指南	GB/T 12260—96	ISO/IEC 9126—91
	计算机软件质量保证计划规范	GB 12504—90	ANSI/IEEE 730
	计算机软件可靠性和可维护性管理	GB/T 14394—93	
	质量管理和质量保证标准 第三部分：GB/T 19001—ISO 9001 在软件开发、供应和维护中的使用指南	GB/T 19000.3—94	ISO 9000—3—93

6. 软件文档

(1) 什么是文档

文档(document)是指某种数据媒体和其中所记录的数据。它具有永久性，并可以由人或机器阅读，通常仅用于描述人工可读的东西。在软件工程中，文档常常用来表示对活动、需求、过程或结果进行描述、定义、规定、报告或认证的任何书面或图示的信息。它们描述和规定了软件设计和实现的细节，说明使用软件的操作命令。

文档是软件产品的一部分，没有文档的软件就不成其为软件。文档的编制在软件开发工作中占有突出的地位和相当大的工作量。高质量、高效率地开发、分发、管理和维护文档对于转让、变更、修正、扩充和使用文档，对于充分发挥软件产品的效益有着重要的意义。

(2) 软件文档的作用

在软件的生产过程中，总是伴随着大量的信息要记录、要使用。因此，软件文档在产品的开发生产过程中起着重要的作用。

§ 提高软件开发过程的能见度。把开发过程中发生的事件以某种可阅读的形式记录在文档中。管理人员可把这些记载下来的材料作为检查软件开发进度和开发质量的依据，实现对软件开发的工程管理。

§ 提高开发效率。软件文档的编制，使得开发人员对各个阶段的工作都进行周密思考、全盘权衡、从而减少返工。并且可在开发早期发现错误和不一致性，便于及时加以纠正。

§ 作为开发人员在一定阶段的工作成果和结束标志。

§ 记录开发过程中的有关信息，便于协调以后的软件、开发、使用和维护。

§ 提供对软件的运行、维护和培训的有关信息，便于管理人员、开发人员、操作人员、用户之间的协作、交流和了解。使软件开发活动更科学、更有成效。

§ 便于潜在用户了解软件的功能、性能等各项指标，为他们选购符合自己需要的软件提供依据。

文档在开发过程中起到了关键作用。从某种意义上来说，文档是软件开发规范的体现和指南。按规范要求生成一整套文档的过程，就是按照软件开发规范完成一个软件开发的过程。所以，在使用工程化的原理和方法来指导软件的开发和维护时，应当充分注意软件文档的编制和管理。

(3) 文档的分类

软件文档从形式上来看，大致可分为两类：一类是开发过程中填写的各种图表，可称之为工作表格；另一类是应编制的技术资料或技术管理资料，可称之为文档或文件。

软件文档的编制，可以用自然语言，特别设计的形式语言，介于两者之间的半形式语言(结构化语言)，各类图形表示。表格来编制文档。文档可以书写，也可以在计算机支持系统中产生，但它必须是可阅读的。

按照文档产生和使用的范围，软件文档大致可分为三类：

① 开发文档：这类文档是在软件开发过程中，作为软件开发人员前一阶段工作成果的体现和后一阶段工作依据的文档。包括软件需求规格说明、数据要求规格说明、概要设计规格说明、详细设计规格说明、可行性研究报告、项目开发计划。

② 管理文档：这类文档是在软件开发过程中，由软件开发人员制定的需提交人员的一些工作计划或工作报告。使管理人员能够通过这些文档了解软件开发项目安排、进度、资源使用和成果等。包括项目开发计划、测试计划、测试报告、开发进度月报及项目开发总结。

③ 用户文档：这类文档是软件开发人员为用户准备的有关该软件使用、操作、维护的资料。包括用户手册、操作手册、维护修改建议、软件需求规格说明。

(4) 软件文档的工作

国家标准局在 1988 年 1 月发布了《计算机软件开发规范》和《软件产品开发文件编制指南》，作为软件开发人员工作的准则和规程。它们基于软件生存期方法，把软件产品从形成概念开始，经过开发、使用和不断增补修订，直到最后被淘汰的整个过程应提交的文档归于以下十三种。下面对其中每一个文档做一些简要的说明：

§ 可行性研究报告：说明该软件项目的实现在技术上、经济上和社会因素上的可行性，评述为合理地达到开发目标可供选择的各种可能的实现方案，说明并论证所选定实施方案的理由。

§ 项目开发计划：为软件项目实施方案制定出的具体计划。它应包括各部分工作的负责人员、开发的进度、开发经费的概算、所需的硬件和软件资源等。项目开发计划应提供给管理部门，并作为开发阶段评审的基础。

§ 软件需求规格说明：对所开发软件的功能、性能、用户界面机运行环境等作出详细的说明。它是用户与开发人员双方对软件需求取得共同理解基础上达成的协议，也是实施开发工作的基础。

§ 数据要求规格说明：给出数据逻辑描述和数据采集的各项要求，为生成和维护系统的数据文件做好准备。

§ 概要设计规格说明：是概要设计工作阶段的成果。它说明系统的功能分配、模块划分、程序的总体结构、输入输出及接口设计、运行设计、数据结构设计和出错处理设计等，为详细设计奠定基础。

§ 详细设计规格说明：着重描述每个模块如何实现，包括实现算法、逻辑流程等。

§ 用户手册：详细描述软件的功能、性能和用户界面，使用户了解如何使用该软件。

§ 操作手册：为操作人员提供该软件各种运行情况的有关知识，特别是操作方法细节。

§ 测试计划：针对组装测试和确认测试，需要为组织测试制定计划。计划应包括测试的内容、进度、条件、人员、测试用例的选取原则、测试结果允许的偏差范围等。

§ 测试分析报告：测试工作完成以后，应当提交测试计划执行情况的说明。对测试结果加以分析，并提出测试的结论性意见。

§ 开发进度月报：该月报是软件人员按月向管理部门提交的项目进展情况的报告。报告应包括进度计划与实际执行情况的比较、阶段成果、遇到的问题和解决的办法以及下个月的打算等。

§ 项目开发总结报告： 软件项目开发完成之后，应当与项目实施计划对照，总结实际执行的情况，如进度、成果、资源利用、成本和投入的人力。此外，还需对开发工作作出评价，总结经验和教训。

§ 维护修改建议：软件产品投入运行之后，可能有修正、更改等问题，应当对存在的问题、修改的考虑以及修改的影响估计等做详细的描述，写成维护修改建议，提交审批。

以上这些软件文档是在软件生存期中，随着各个阶段工作的开展适时编制的。其中，有的仅反映某一个阶段的工作，有的则需跨越多个阶段。表 10.2 给出了各种文档应在软件生存期中哪个阶段编写。表 10.3 说明哪些人与哪些文档的编制有关。

表 10.2 软件开发项目生存期各阶段与各种文档编制工作的关系

阶段 / 文档	可行性研究与计划	需求分析	软件设计	编码与单元测试	集成与测试	运行维护
可行性研究报告	←					
项目开发计划	←	←				
软件需求规格说明		←				
数据要求规格说明		←				
测试计划		←	←			
概要设计规格说明			←			
详细设计规格说明			←			
用户手册		←	←	←		
操作手册			←	←		
测试分析报告					←	
开发进度月报	←	←	←	←	←	
项目开发总结					←	
程序维护手册（维护修改建议）						←

表 10.3 人员与文档关系

文档	管理人员	开发人员	维护人员	用户
可行性研究报告	■	■		
项目开发计划	■	■		
软件需求规格说明		■		
数据要求规格说明		■		
测试计划		■		
概要设计规格说明		■	■	
详细设计规格说明		■	■	
用户手册				■
操作手册				■
测试分析报告		■	■	
开发进度月报	■			
项目开发总结	■			
程序维护手册（维护修改建议）	■		■	

(5) 对文档编制的质量要求

为使软件文档能起到多种桥梁的作用，使它有助于程序员编制程序，有助于管理人员监督和管理软件的开发，有助于用户了解软件的工作和应做的操作，有助于维护人员进行有效的修改和扩充，文档的编制必须保证一定的质量。

高质量的文档应当体现在以下几个方面

① 针对性：文档编制以前应分清读者对象。按不同的类型、不同层次的读者，决定怎样适应他们的需要。例如，管理文档主要面向管理人员，用户文档主要面向用户，这两类文档不应像开发文档(面向开发人员)那样过多使用软件的专用术语。

② 精确性：文档的行文应当十分确切，不能出现多义性的描述。同一课题几个文档的内容应当是协调一致，没有矛盾的。

③ 清晰性：文档编写应力求简明，如有可能，配以适当的图表，以增强其清晰性。

④ 完整性：任何一个文档都应当是完整的、独立的，它应自成体系。例如，前言部分应做一般性介绍，正文给出中心内容，必要时还有附录，列出参考资料等。同一课题的几个文档之间可能有些部分内容相同，这种重复是必要的。不要在文档中出现转引其它文档内容的情况。例如，一些段落没有具体描述，而用“见××文档××节”的方式，这将给读者带来许多的不便。

⑤ 灵活性：各个不同软件项目，其规模和复杂程度有着许多实际差别，能一律看待。

§ 应根据具体的软件开发项目，决定编制的文档种类。软件开发的管理部门应该根据本单位承担的应用软件的专业领域和本单位的管理能力，制定一个对文档编制要求的实施规定。主要是：在不同条件下，应该形成哪些文档？这些文档的详细程度？该开发单位每一个项目负责人都应当认真执行这个实施规定。

§ 当所开发的软件系统非常大时，一种文档可以分成几卷编写。例如，项目开发计划可分写为：质量保证计划、配置管理计划、用户培训计划、安装实施计划等。系统设计规格说明可分写为：系统设计规格说明和子系统设计规格说明。程序设计规格说明可分写为：程序设计规格说明、接口设计规格说明和版本说明。操作手册可分写为：操作手册和安装实施过程。测试计划可分写为：测试计划、测试设计规格说明、测试规程、测试用例。测试分析报告可分写为：综合测试报告、验收测试报告。项目开发总结报告也可分写成：项目开发总结报告、资源环境统计。

§ 应根据任务的规模、复杂性、项目负责人对该软件的开发过程及运行环境所需详细程度的判断，确定文档的详细程度。

§ 对国标 GB8567－88《计算机软件产品开发文件编制指南》所建议的所有条款都可以扩展，进一步细分，以适应需要；反之，如果条款中有些细节并非必需，也可以根据实际情况压缩合并。

§ 程序的设计表现形式，可以使用程序流程图、判定表、程序描述语言(PDL)、或问题分析图(PAD)等。

§ 对于文档的表现形式，没有规定或限制。可以使用自然语言、也可以使用形式化的语言。

§ 当国标《计算机软件产品开发文件编制指南》中所规定的文档种类不能满足某些应用部门的特殊需要时，可以建立一些特殊的文档种类要求。这些要求可以包含在本单位的文档编制实施规定中。

⑥ 可追溯性：由于各开发阶段编制的文档与各个阶段完成的工作有密切的关系，前后两个阶段生成的文档，随着开发工作的逐步延伸，具有一定的继承关系，在一个项目各开发阶段之间提供的文档必定存在着可追溯的关系。例如，某一项软件需求，必定在设计说明书、测试计划、甚至用户手册中有所体现。必要时应能做到跟踪追查。

7. 软件过程 CMM 标准与过程改进

软件过程是软件生存期中的一系列相关的过程，又称为软件生存期过程。过程是活动的集合，活动是任务的集合。任务是将输入变换为输出的操作。活动的执行可以是顺序的，重复的，并行的、嵌套的。

软件过程的考虑主要针对软件生产和管理。为了得到满足要求的软件产品，不但需要有好的开发方法，还需要有好的工程支持和工程管理。就是说，软件过程不仅要有工程观点，还应有系统观点、管理观点、运行观点和用户观点。

软件过程有多种分类。按照不同人员的工作内容来分，国际标准化组织在 ISO／IEC 12207－1995 中将软件过程分为 3 类：基本过程、支持过程和组织过程。并且当把软件过程运用到相关组织，运用到具体应用论域和具体项目中时，可以根据特定情况。对各种过程和活动进行剪裁，形成所需要的软件过程模型。

(1) ISO／IEC 12207－1995 概要

图 10.16 给出了软件生存期的各种过程，其主要活动和使用者见表 10.4。

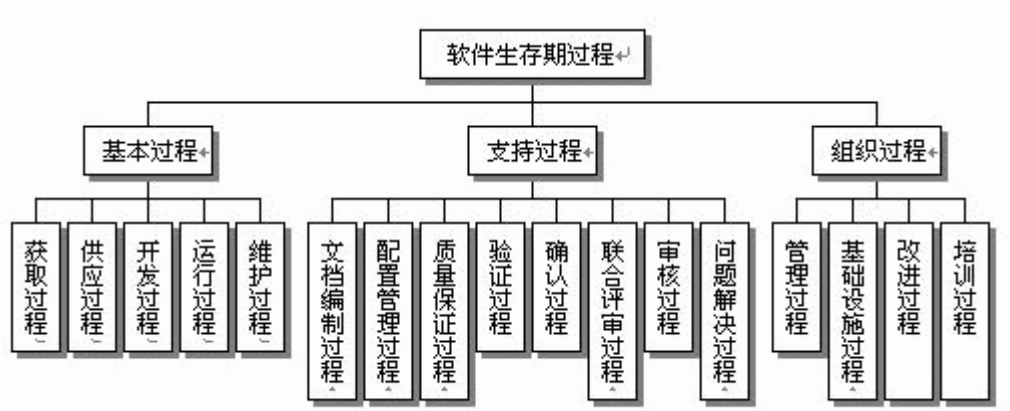


图 10.16 软件过程

表 10.4(a) 基本过程

1	获取过程(需方)	(是需方为了获得一个软件产品所进行的一系列活动) 分析，确定，提出需求，准备招标，准备合同，监督供方，验收
2	供应过程(供方)	(是供方为向需方提供软件产品所进行的一系列活动) 理解需求，准备投标，签定合同，制定及实施计划，评审及评价，交付
3	开发过程(开发者)	(是软件开发者根据合同开发和交付软件的一系列活动) 准备实施：选择活动，制定计划，制定文档编制方式，安排支持过程 系统需求分析，系统结构设计 软件需求分析，体系结构设计，详细设计，编码，集成测试，确认测试 系统集成，系统确认，软件安装，软件验收支持
4	运行过程(运行者)	(在软件开发过程完成后，将该软件从开发环境转移到用户的实际运行环境。在运行时对用户的要求提供帮助和咨询，并对运行效果进行评价) 准备实施：准备运行环境，制定运行过程实施计划和运行评价标准 系统运行，运行测试，对用户提供支持，系统运行评价，用户运行评价
5	维护过程(维护者)	准备实施：确定接受和反馈修改要求步骤，制定各支持过程实施方法 问题和变更分析，实施变更，维护评审和验收，移植，软件退役

表 10.4(b) 支持过程

1	文档编制过程	(是一个记录由某一过程或活动所产生的信息的过程) 制定文档编制计划，设计文档，制作文档，编辑，评审，批准，发行，维护
2	配置管理过程	制定配置管理计划，配置标识，版本控制，变更控制，配置报告，配置审核，发行管理和提交
3	质量保证过程	(是一个为使软件过程和软件产品符合规定需求，并按预定计划按时完成提供适当保证的过程) 过程剪裁，制定过程实施计划，产品、过程的质量保证
4	验证过程	(确定系统或软件的需求是否完备和正确，以及每一阶段的软件产品是否达到前一阶段对它的要求和条件) 制定验证计划，合同、过程、需求、设计、代码、集成、文档的验证
5	确认过程	(确保最终产品满足预期的使用要求) 制定确认计划，准备测试需求、测试用例、测试规格说明 实施测试，确认度量标准，确认测试结果、软件产品用途的适用性
6	联合评审过程	(评审方与被评审方共同对某一活动或阶段的执行情况和产品进行评审) 项目管理评审，技术评审
7	审核过程	按项目实施计划定期举行，审核项目是否按需求、计划、合同完成 审核产品、测试数据和验收，审核测试报告和用户手册的完整性和适用性 审核进度和成本
8	问题解决过程	(分析和排除在开发、运行、维护或其它过程中发现的问题和不一致) 问题分类、排优先次序，启动有关活动指明问题原因，分析和解决问题 评价解决结果，确定是否引入新的问题

表 10.4(c) 组织过程

1	管理过程(管理者)	(管理包括进度管理、成本管理、质量管理、人员管理、资源管理、标准化管理，管理对象是进度、系统规模及工作量估算、经费、组织机构很人员、风险、质量、作业和环境配置等) 制定计划，实施和监控计划，评审和评价计划的完成程度，编写管理文档
2	基础设施过程	(建立、维护各个过程所需的基础设施的过程) 基础设施包括硬件、软件、工具、技术、标准，以及开发、运行、维护所需的设施
3	改进过程	(建立、评估、度量、控制和改进软件生存期过程的过程) 制定组织计划，评估相关过程，实施分析，改进过程
4	培训过程	(为系统或软件产品提供人员培训的过程) 制定用人计划和培训计划，开发培训资料，实施培训活动

(2) 剪裁过程

为有效地实施软件过程，需要针对特定应用论域的软件工程，剪裁选定的软件过程模型和标准，以形成该工程的模型和标准，形成该工程的各个软件过程和活动。

剪裁过程是对软件过程和活动实施剪裁的过程。其主要活动有：

① 指明工程环境

指明影响剪裁的工程环境特征。如使用的工程模型和方法，软件需求，机构的政策和策略，软件规模、重要性和类型，参与人员和合作者的素质及人数。

② 收集信息

向所有可能会影响剪裁的组织或个人收集信息。参与人员包括用户、工程支持人员、负责签定合同签定的人员、潜在的投标者。

③ 选取过程、活动和任务

根据收集到的信息，确定要实施的过程、活动和任务。

④ 编制文档

将所有的剪裁决策和作出这些决策的理由记入文档。

剪裁可分为两级：第一级，根据不同的应用论域进行剪裁；第二级，根据每一个具体项目或合同进行剪裁。

(3) 过程模型建造技术

软件过程建模是对实际软件过程的再工程。过程建模活动是软件过程中最主要的活动，所有其它的工程活动都是基于建模活动的结果进行的。过程建模技术包括建模目的、建模方法、建模语言、软件过程与过程模型的度量。其中，建模目的决定了建模活动的范围；建模方法和建模语言是对建模目的的支持和将过程模型形式化的手段，并决定了过程模型的质量；对过程模型的度量与评估是修改和演进过程模型的基础，使得过程模型能够更好地与软件工程相吻合，并充分地反映一个组织的过程成熟度。因而，建模方法与语言是过程建模技术的关键。

① 过程建模的目的

§ 使人们易于理解软件过程并在此基础上进行交流。通过过程建模，可以使人们对过程达到一个共识，共享过程知识，并在此基础上进行交流。

§ 支持对过程的分析。通过对过程进行形式化或非形式化的建模，人么可以对过程活动和过程之间的相互关系进行分析、比较和预测，以评估和改善过程的有效性。

§ 支持过程中的通信。过程模型可以为有关人员、小组和项目提供必要的过程信息和通信支持，使人们之间的协同工作更加有效。

§ 支持过程的改进。对过程模型各部分的功效进行分析，找出可以改善的部分，对模型进行修改，并可在模型实际修改前估计修改产生的影响。从而支持严格管理下的过程进化。

§ 支持过程的管理。过程模型可以帮助人们制定项目计划，监控、管理和协调项目的实施过程，估算软件创建或演进时各活动的进展。过程模型还可以成为过程度量的基础。

§ 支持过程的复用。通过过程模型可以复用定义良好的软件过程。可以针对具体项目的特点，对已有的过程进行剪裁或扩充，使之适合特定项目的需要。

§ 提供对过程的自动执行支持。这需要一个基于过程驱动的软件工程环境。支持包括对用户的资源配置、提供各用户间的通信服务，组织和协调个人或小组间的工作等。

② 过程建模方法

过程建模涉及到软件产品的开发和维护、软件项目管理、过程管理和过程改进等各个方面，涉及到过程的活动、角色、产品、资源和约束等各种过程实体，还涉及到建模所用的形式化方法，加上软件过程本身的复杂多样性，从而使得构造过程模型的方法也是多种多样的。考虑过程所涉及的实体和建模所采用的不同形式化方法，可以有两种不同的分类方法。

§ 考虑过程所涉及实体的分类方法

ⅰ）以活动为中心的建模方法

首先考虑过程活动以及它们之间的执行顺序，再收集与各个活动相关的其它数据，如活动所涉及的角色、产品、资源和约束等，从而建立过程模型。这种方法能够直观地反映实际过程的工作流程，且无二义性，使人们可以很容易地理解、分析、计划、管理和控制过程的执行。缺点是，由于活动是动态实体，且实际流程也是动态多变的，因此，过程缺乏稳定性。实际过程中发生的任何变化都将影响原来的过程模型。此外，要求建模人员应具备所建模型的各种过程活动的知识，并在模型执行时进行一致性维护。这种模型适合于对实际过程的指导和控制。

ⅱ）以角色为中心的建模方法

首先确定各个角色的任务和角色之间的关系，再以角色为中心收集过程的其它数据，如活动、产品、资源和约束等，建立过程模型。由于角色是组织结构中的基本构成因素，是一个易于理解和接受的相对稳定的抽象实体，同时一个项目的任务通常是按角色进行分解的，所以，以角色为中心的缄默方法比较自然。而且角色是过程的一个不变的实体，因此，构造的过程模型比较稳定。这种方法还能够

明确地描述过程的组织方面的信息，使得参与项目的人们易于明确自己的任务，便于对项目的计划、管理和控制。缺点是不能对过程的工作流程有一个显式的描述和定义，人们难以从整体上把握一个过程和他们在过程中的位置。另外，随着任务的逐步分解、细化，涉及的角色增多，角色之间的关系更加复杂，不利于低层次上的过程管理。 除了以上两种建模方法以外，还有以产品为中心的建模和基于过程模板的建模。

§ 考虑过程所采用的形式化方法的分类方法

ⅰ）过程程序设计方法

这种建模方法的出发点是“软件过程也是软件”。Osterweil 认为软件过程与软件产品具有广泛的类同性，对软件过程的描述也是一种程序设计形式。这种方法通过关系、谓词和触发器等机制对软件过程的功能、行为和对象进行详细、确定的算法描述。这种方法不是控制活动执行的方法，而是控制生产产品的交互过程。其优点是可以成功地进行交互控制，自动维护对象之间的一致性。缺点是严格过程化，不能支持大范围的兵法活动，无法描述软件过程的动态变化情况。

ⅱ）功能分解方法

这种建模方法用一组反映输入／输出关系的过程元素(数学函数)来表示软件过程。这组函数可以按照语法进一步层次分解，形成一个过程的多个子过程步。这种分解可以一直进行下去，直到产生的子过程步可以映射到一个外部工具或由人操作的过程为止。这种方法支持子过程步的并行执行、串行执行、迭代执行，还提供了控制过程状态行为的元操作，如创建、挂起、恢复执行等。

ⅲ）基于 Petri 的建模方法

在当前的过程建模方法中，基于 Petri 网及其变种的建模方法较多。因为 Petri 网能够有效地形式化描述软件过程的并发性和活动与产品之间的关系，而且使用这种图形表示描述软件过程，易于理解和管理。其优点是较好地考虑了任务的激活条件、活动的执行顺序、活动产生的信息实体之间的转换情况。缺点是忽视了活动对内部状态所产生的影响。，另外，由于网络的全局相关性，使得对过程的任何改动都会影响到其它部分，不利于过程复用。

ⅳ）基于规则的建模方法

基于规则的建模方法和语言是人们普遍看好的一种过程建模技术。这种方法提供了活动的动态链接机制，很自然地描述了过程的不可预见性，为人们控制过程提供了非常灵活的手段。基于规则的建模语言通常可提供回溯、向前链接、向后链接等自动执行机制，还可提供规则推理、调度和控制过程活动的机制，并可灵活方便地修改过程。其缺点是不利于人们理解软件过程，构造、分析过程模型时也比较困难，缺乏对并行工作与协同工作的支持。

ⅴ）基于知识的建模方法

基于知识的建模方法提供了对过程模型的增量式形式说明能力和可复用能力。这种方法使用面向对象的技术把过程知识(如过程活动、过程实施者、产品对象和工具以及它们之间的关系等)抽象成不同的类，存放于知识库中。过程建模时，根据要求查询知识库，获取有关过程活动及其它成分的抽象描述，从中选取或构造所需的过程模型，并对其进行分析和推理，最后生成过程实例和相应的活动计划。用这种方法构造的过程模型是活动的类的层次结构，其中每个活动类与子类都对应有多种资源需求，如要加工的数据、所需工具、开发角色等。用类的继承关系描述可以表示各种过程关系，如控制流关系、任务的前置条件和后置条件、不同角色之间的上下级关系、产品的组成关系等。

一般来讲，建模是否成功，要看所建立的模型是否达到了建模目的。为了使建模方法具有通用性和适应能力，可以采用混合风格的建模方法，将多种风格形式化于一身。但这种方法要很好地处理各个模型成分之间的转换、通信、协调和相互作用等问题。

(4) 软件过程的改进

1983 年美国 TRW 公司 B.W.Boehm 提出著名的软件工程七条原则：

§ 按软件生存期分阶段制定计划并认真实施；

§ 逐个阶段进行确认；

§ 坚持严格的产品控制；

§ 适用现代程序设计技术；

§ 明确责任；

§ 用人少而精；

§ 不断改进开发过程。

由此可知，不断改进软件开发过程是软件工程的基本原理之一。

实践表明，软件过程需要不断完善。首先从非工程化的软件开发方式转变为工程化的软件开发方式，按照软件工程的系统方法进行软件的工程活动和管理活动，进而不断完善各个软件过程，不断提高软件过程能力。随着这种能力的提高，一个软件机构在完成软件产品时在预算、进度，特别是产品质量方面的风险不断降低。因此，软件过程能力的提高需要首先对当前的软件过程状况进行评估。 当前，在过程评估和过程改进方面已经做了很多工作，并制定出了具有指导意义的标准。表 10.5 列出了目前著名的几种软件过程评估和过程改进方案。

表 10.5 几种软件过程评估和软件过程改进方案

推出机构	名称	时间	主题
CMU－SEI 美国卡内基·梅隆大学软件工程研究所	CMM (Capability Maturity Model)	1987	软件过程能力成熟度模型
ISO 国际标准化组织	ISO 9001, ISO 9000－3	1994, 1997, 2000	建立和维持质量体系
ISO/IEC 国际标准化组织/国际电工委员会	ISO/IEC TR15504 SPICE (Software Process Improvement and Capability dEtermination)	1997	软件过程改进和能力评估
DOD 美国国防部	MIL－STD－498	1984	软件过程改进
Bell 加拿大贝尔公司	TRILLTUM		软件过程评估(基于CMM)
德国联邦武装部队	V－Model	1992	软件过程定义
BOOTSTRAP Institute 欧共体	BOOTTSTRAP	1994	软件过程评估(基于CMM)

无论哪一种软件过程改进方案的实施都要从已有的软件过程开始，改进工作需经历一系列步骤。图 10.17 描述了这些步骤所构成的软件过程改进模式。

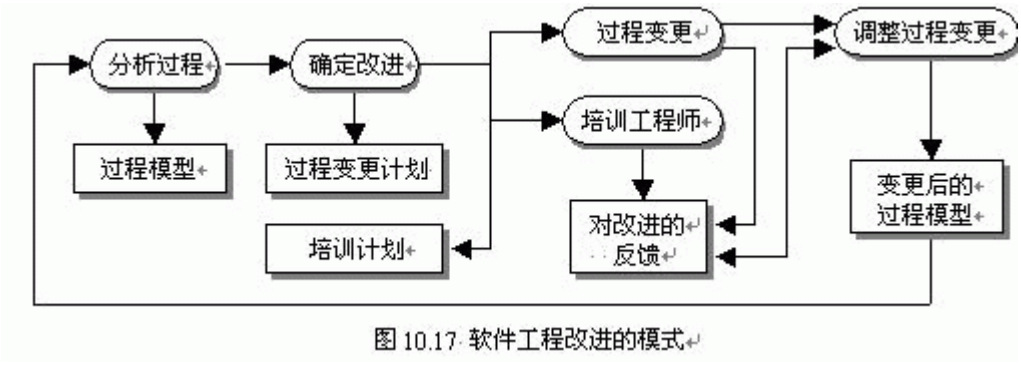


图 10.17 软件工程改进的模式

① 过程分析：考察和理解现有的过程。在一些情况下需要对过程的某些环节进行度量和定量分析，利用取得的数据来表明过程的状况。同时，这些数据可以用来与过程改进后的状况进行对比。

② 确定改进：利用过程分析的结果，找出原有过程中质量、进度和成本的瓶颈。针对发现的问题，制定过程改进方案，提出需要采用什么规程、方法和工具的建议。

③ 过程变更：实施过程变更，把新的规程、方法和工具安置于合适的过程环节上，并且与其它的软件过程活动集成起来。需要注意的是，要保证变更与其它的过程活动及更为广泛适用的机构规程和标准相协调。

④ 培训：没有培训的过程变更在大多数场合注定要失败。有的单位在培训工作不够充分的情况下，强制推行过程变更，这样做也不会收到好的效果。

⑤ 调整过程变更：在初步实施过程变更后，不可能立即收到完满的效果，在过程修改后还可能会出现一些小的问题，这就需要进行适当的调整。

过程改进的步骤需要反复进行。在完成调整后，又可能要返回去重新进行过程分析，再执行其后续步骤。此外，同时引入太多的变更是不切实际的。

8. 软件过程能力评估的 CMM 模型

(1) 软件机构的过程成熟度

软件开发的风险之所以大，是由于软件过程能力低。其中最关键的问题在于软件开发机构不能很好地管理其软件过程，从而使得一些好的开发方法和技术起不到预期的作用。当然，即使是在这样的机构中，个别软件项目仍能产生高质量的产品，但这是通过特定优秀软件人员的努力，而不是通过重复使用具有成熟软件过程的方法。在没有全机构范围的软件过程的情况下，能否继续成功地开发下一个项目，完全取决于能否留住这些优秀的软件人才。仅仅建立在特定人员上的成功不能为全机构的生产率和质量的长期提高打下基础，必须在建立有效的软件工程实践和管理实践的基础上，坚持不懈的努力，才能不断改进。

对于不同的软件开发机构，在组织人员完成软件项目中所依据的管理策略有很大差别，因而软件项目所遵循的软件过程也有很大差别。在此，我们用软件机构的成熟度(Maturity)加以区别。表 10.6 给出不成熟的软件机构和成熟的软件机构的比较。

表 10.6 不成熟的软件机构和成熟的软件机构的比较

	不成熟的软件机构	成熟的软件机构
软件过程	由参与开发的人员临时拼凑。有时即使确定了，实际上并不严格执行。	建立了软件开发和维护过程。人们对其有较好理解。一切活动均遵循过程的要求进行，做到工作步骤有次序，且有章可循。
管理方式	反应型：管理人员经常要集中精力去应付难以预料的突发事件。	主动型：软件过程不断改进，产品质量和客户满意程度负责质量保证的经理负责监控。
进度、经费估计	估计不切实际。在进度拖延情况下，不得不降低软件的质量。	根据以往项目取得的实践经验确定，因而比较符合实际情况。
质量管理	产品质量难以预测。质量保证活动，如质量评审、测试等，常被削弱或被取消。	产品质量有保证，软件过程有管理，具有必要的支持性基础设施。

在各个软件机构的过程成熟度有着相当大的差别面前，为了做出客观、公正的比较，需要建立一种衡量的标尺。使用这个标尺可以评价软件承包机构的质量保证

能力，在软件项目评标活动中，选择中标机构。另一方面，这一标尺也必然成为软件机构改进软件质量，加强质量管理，以及提高软件产品质量的依据。

(2) 软件机构的能力成熟度模型 CMM（Capability Maturity Model）

1987 年美国卡内基－梅隆大学软件工程研究所(SEI)受美国国防部资助，提出了软件机构的能力成熟度模型 CMM,经过几年的使用及 1991 年和 1993 年两次修改，现已成为具有广泛影响的模型。CMM 将软件过程的成熟度分为 5 个等级，如图 10.18 所示。

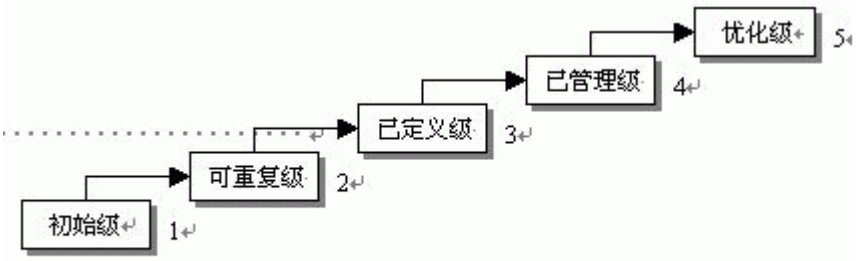


图 10.18 软件过程成熟度模型

表 10.7 各个等级的软件机构的特征

级别	1. 初始级	2. 可重复级	3. 已定义级	4. 已管理级	5. 已优化级
特点	▪ 过程执行杂乱无序。 ▪ 管理无章。 ▪ 开发项目成效不稳定，产品的性能和质量依赖于个人能力和行为。	▪管理制度化，工作有章可循。 ▪ 开发工作初步实现标准化。 ▪ 变更基线化。 ▪ 过程可跟踪。 ▪ 新项目计划和管理基于过去实践经验，具有重复以前成功项目的环境和条件。	▪开发过程标准化、文档化。 ▪ 完善的培训和专家评审制度。 ▪ 技术和管理活动稳定实施。 ▪ 项目质量、进度和费用可控制。 ▪ 项目过程、岗位和职责均有共同的理解。	▪ 产品和过程有定量质量目标。 ▪ 过程的生产率和质量可度量。 ▪ 有过程数据库。 ▪ 实现项目产品和过程的控制。 ▪ 可预测过程和产品质量趋势，如预测偏差，及时纠正。	▪ 不断改进过程，采用新技术、新方法。 ▪ 有防止缺陷，识别薄弱环节及加以改进的手段。 ▪ 可通过反馈取得过程有效性的统计数据并可据此进行分析，改善过程。

(3) 跳跃成熟度等级的错误

处于较低等级的软件机构可以而且往往需要事实较高等级上的某些过程，例如，CMM 中在等级 3 以前不讨论软件产品工程过程的活动，如需求分析、设计、编码、测试，但实际上处于等级 1 的软件机构都必须进行这些活动。又例如，处于等级 1 或等级 2 的机构可以组织同行专家评审，但这是属于等级 3 的。但是否可以得出一个结论：处于等级 1 的软件机构可以直接实施等级 3 的一些要求，从而它就可提升到等级 3 呢？这是不可以的。

跳跃等级是违反发展规律的。每个等级形成一个必要的基础，从此基础出发才能达到下一个等级。CMM 划分 5 个等级，只是列出了在一个等级上占主导地位的问题。一个机构必须也必然逐步经历这些等级才能建立起优秀的软件工程文化。假如一个管理混乱的软件机构试图实施过程优化(等级 5),由于没有可定量度量和跟踪的手段,对过程变更后可能产生的后果缺乏了解,这种过程优化最终会失败。

总而言之，软件能力成熟度等级的提高是一个循序渐进的过程，具有实施较高成熟度等级的某些活动的能力，并不表明可以跳跃成熟度等级。

(4) 关键过程域 KPA（KPA，Key Process Area）

除去初始级以外，其它 4 个等级都有若干个指导软件机构改进软件过程的要点，称为关键过程域。每一个关键过程域是一组相关的活动，且这些活动都有一些达标的标准，用以表明每个关键过程域的范围、边界和意图。为达到关键过程域的目标所采取的手段可能因项目而异，但一个软件机构为实现某个关键过程域，必须达到该关键过程域的全部目标。只有一个机构的所有项目都已达到某个关键过程域的目标，才能说该软件机构的以该关键过程域为特征的过程能力规范化了。

表 10.8 给出了各成熟度等级对应的关键过程域和主要工作。

表 10.8 各成熟度等级对应的关键过程域

级别	1. 初始级	2. 可重复级	3. 已定义级	4. 已管理级	5. 已优化级
关键过程域		• 需求管理 • 软件项目策划 • 软件项目跟踪和监控 • 软件子合同管理 • 软件质量保证 • 软件配置管理	• 软件机构过程焦点 • 软件机构过程定义 • 培训大纲 • 集成软件管理 • 组间协调 • 软件产品工程 • 同行专家评审	• 定量过程管理 • 软件质量管理	• 过程变更管理 • 缺陷预防 • 技术变更管理
主要工作	•过程活动杂乱无序 •开发过程的可重复性差	• 客户与软件项目间对客户要求有共同理解 • 制定软件工程和软件管理的合理的计划 • 建立适当的对实际进展的跟踪和监控 • 选择合格的软件承包方，并有效管理 • 提供对软件项目所采用的过程和产品质量的适当的可视性 • 需求变更和产品基线控制	• 规定软件机构在提高整体过程能力，改进软件过程活动方面的责任 • 开发和维持一批便于使用的软件过程财富 • 培训个人的技能和知识，以高效执行其任务 • 根据项目的要求裁剪和优化，将软件工程活动和管理活动集成为一个协调的定义良好的软件过程 • 制定组间合作的方法 • 一致地执行妥善定义的软件工程过程 • 通过设计评审、结构化走查或其它学院式评审方法实施同行评审	• 为已定义的过程建立一套详细的性能度量机制 • 为产品和过程设立质量目标，度量软件过程和产品	• 用第 4 级建立的度量机制，不断地改进软件机构中的软件过程 • 识别缺陷出现的原因，防止它们再次出现 • 识别能带来好处的新技术，以有序的方式引进这些新技术，能在不断变化的环境中高效率地创新

(5) 关键实践

关键实践是对关键过程域起关键作用的方针、规程、措施、活动以及相关基础设施的建立。为了达到关键过程域所规定的目标，必须实施相应的关键实践。每个关键过程域所包含的关键实践涉及 5 个方面：执行约定、执行能力、执行的活动、测量和分析、验证实施，人们统称为 5 个共同特征。关键过程域所包含的关键实践全部按这 5 个共同特征加以组织，如图 10.19 所示。

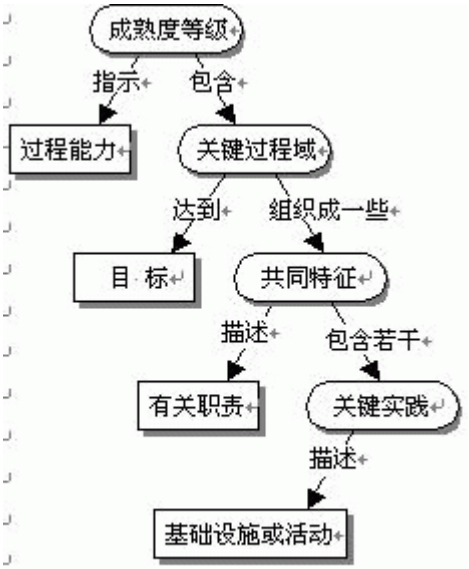


图 10.19 CMM 的结构

共同特征是表明一个关键过程域的实施和规范化是否有效、可重复且持久的一些属性。5 个共同特征的含义如下。

§ 执行约定：描述一个机构在保证将过程建立起来并持续起作用方面所必须采取的行动。执行约定一般包括制定机构的方针和规定高级管理人员的支持。

§ 执行能力：描述为了实施软件过程，项目或机构中必须存在的先决条件。执行能力一般包括资源、组织机构和培训。

§ 执行的活动：描述为了实现一个关键过程域所必须的角色和规程(即描述必须由谁做什么)。执行的活动一般包括制定计划与规程、执行计划、跟踪执行情况，必要时采取纠正措施。

§ 测量和分析：描述对过程进行测量和对测量结果进行分析的需要。测量和分析一般包括一些为了确定所执行活动的状态及有效性所能采用的测量和分析。

§ 验证实施：描述保证遵照已建立的过程进行活动的措施。验证一般包括管理人员和软件质量保证部门所做的评审和审核。

国际上有一个公认的基本观点是：整个软件过程的改进是基于许多小的，进化的步骤，而不是通过一次革命性的创新来实现的。这些小的进化步骤就是通过关键实践来实现的。

(6) 软件人员能力成熟度模型 为了提高软件机构的生产能力，必须关注三个相关的因素：技术、过程和人员。软件人员能力成熟度模型采用了软件过程成熟度模型 CMM 的框架，帮助软件机构了解其劳动力管理活动的成熟度，将软件过程改进与劳动力管理的改进结合起来，指导软件机构管理和开发其劳动力资源。建立该模型的战略是：

§ 通过提高劳动力的能力来提高软件机构的能力；

§ 确保软件开发能力属于机构而不是个别人；

§ 使员工的个人动机与机构的动机保持一致；

§ 使机构能够留住人才(尤其是关键人才)。

软件人员能力成熟度模型(P-CMM)最早是由 CMM 小组成员 Bill Curtis 于 1990 年提出来的，1995 年 SEI 正式公布了 P-CMM 的 V1.0 版。

作为成熟度模型，它是一个分级进化的模型，以作为机构循序渐进的指南。每一个成熟级别，既是机构发展的阶段性目标，又是评价机构能力水平的一个标准。

P-CMM 包含了 5 个成熟度级别，如图 10.20 所示。

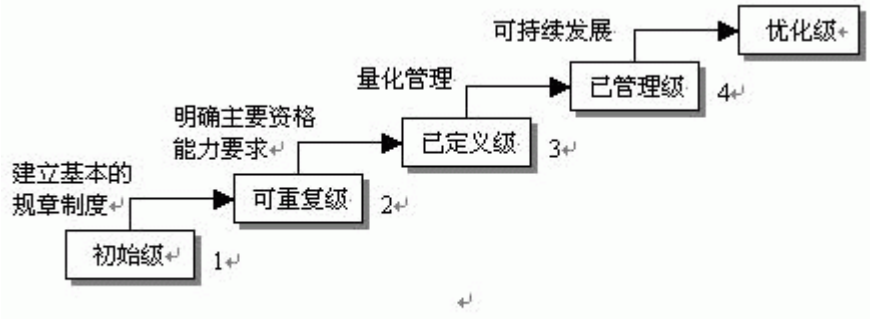


图 10.20 软件人员能力成熟度模型的框架

① 成熟度级别：是一个恰当定义的进化平台，反映软件机构在管理和开发人力资源能力方面所达到的水平。

② 劳动力能力：软件开发机构中劳动力的知识和技能水平，以及劳动力用这些知识和技能提高业务工作绩效的能力。它直接决定企业的业绩和企业实现其商业目标的能力。

③ 关键过程域：除了初始级，每个成熟度级别都可以由一组关键过程域组成，指明为达到该成熟度级别，机构应致力于在这几个领域内改善其工作。每一关键

过程域又与一组目标相连接。当达到这些目标时，表明软件机构建立了该领域的劳动力的能力。

表 10.9 给出了 P-CMM 各成熟度等级对应的关键过程域和主要工作。

表 10.9 P-CMM 各成熟度等级对应的关键过程域

级别	1：初始级	2：可重复级	3：已定义级	4：已管理级	5：已优化级
关键过程域		·工作环境 ·交流与沟通 ·人员组织 ·业绩管理 ·培训 ·薪酬	·知识与技能分析 ·劳动力规划 ·业务能力的开发 ·职业发展 ·基于资格能力的条例 ·建立参与和分享的企业文化	·业务指导 ·团队建设 ·基于团队的条例 ·企业级的业务能力管理 ·企业级的业绩整合	·个人业务能力的发展 ·教练 ·持续的劳动能力的创新
主要工作	·主管人员未经训练 ·员工业绩评估无计划 ·引进报酬制度未经效率分析 ·不适当决策会引起员工不满 ·员工个人动机未与企业动机结合导致有更好工作环境或发展机会就会跳槽	·建立和保持必须的工作环境，排除干扰 ·建立有利于沟通的氛围，确保上下通信渠道畅通 ·建立和实施一套招聘、选拔、任命等人事制度 ·建立业绩与标准体系和业绩汇报制度 ·明确培训需求，加强培训以保证员工掌握完成工作所需技能 ·根据员工对机构的价值和所做贡献来提供报酬	·明确企业关键业务过程所需的知识和技能，维持一定的知识和技能储备 ·为整个企业的劳动力能力发展建立目标和规划 ·制定培训计划或其它发展规划，从整体上提高企业的劳动力能力 ·鼓励员工，为员工学习新技能、增长才干和追求更高目标提供机会 ·保证所有劳动条例都与劳动力知识与技能性能 ·在机构的各个层次上建立有效的通信渠道，使员工了解决策过程，并将决策结果告知员工	·指定企业内有经验的人员作为个人或小组的业务指导 ·最大限度地练成并利用团队中不同的知识和技能为企业业务发挥作用 ·调整劳动力条例使其有利于促进团队的发展 ·增强企业关键业务的资格能力，建立资格能力增长的定量指标，定期收集数据，分析改进效果，利用分析结果指导有关工作的改进 ·建立个人、团队、部门、企业各级可度量目标，定期收集数据，分析发展趋势，处理例外情况，总结成果	·为员工在业务上的自我发展奠定基础 ·选择合适的专家为指导，为个人或团队的业绩提高进行方法指导，评价业务进步情况 ·评价和确认有效的劳动力条例并保证其在全企业的贯彻执行，鼓励合理化建议，引进新的条例并进行探索性的试验，推广有益的革新性措施

④ 关键实践：每个关键过程域由关键实践构成，它是为保证关键过程域内容的有效实施和制度化而建立的基本活动。关键实践规定了在该领域的基本策略、步骤和活动。注意，关键实践只规定了做什么，不规定怎么做。

⑤ 劳动力条例：在劳动力与员工管理领域中，为贯彻和实施企业内部政策而建立的程序和指南。如薪酬、业绩管理、团队建设、培训等方面的规定。

⑥ 目标：目标概括了当关键过程域被有效而长久地实施后应达到的状态。完成目标的程度表明了机构在该成熟度级别上达到了何种能力。

⑦ 共同特征：共同特征强调关键过程域的实施和制度化，关键过程域中的关键实践按共同特征分为 5 类。这 5 种共同特征是：

§ 实施保证：涉及机构内部政策的制定和领导责任的确定。

§ 实施能力：描述先决条件、资源保证、组织结构和人员培训。

§ 实施活动：描述关键过程域的实施步骤和角色需要。一般包括工作规划、操作程序、具体实施、跟踪与调整。

§ 度量与分析：建立可操作的度量用例，以判定实施活动的状态和有效性。

§ 实施审核：检查所实施的活动是否符合已建立的政策和程序，涉及有关方面的审查和审核活动。

以上的 5 种特征中，以实施活动的比重最大，因为只有它具体描述了为建立劳动力能力所需要的实施过程。

⑧ 软件人员能力成熟度模型的结构：按照 SEI 的定义，根据成熟度级别框架展开的 P-CMM 模型结构如图 10.21 所示。

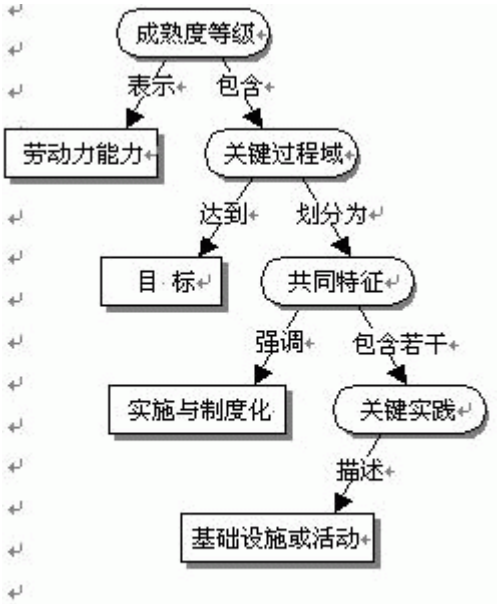


图 10.21 P-CMM 结构

P-CMM 模型由 5 个成熟度级别构成，每个级别代表企业所达到的劳动力能力水平，每一级包含了若干个关键过程域，每一个关键过程域由一组相应的目标。为达到这些目标，应采取一系列基本措施和活动，这就是关键实践。应将这些关键实践的内容制度化，以保证其有效、长久地执行。为此，可用 5 种共同特征来概括和组织它们。根据不同侧重，这些特征中既有先决条件保证，又有具体实施操作程序，还有实施后的审查和审核等，从而保证了每个关键过程域目标的实现，进而

实现向某一成熟度级别的进化。这样一个模型是可操作的，它对软件机构人员能力开发工作具有很大的指导意义。

⑨ 主题：关键过程域是在每个级别上定义的，但是在跨越不同级别的关键过程域之间还存在着纵向的联系，这就是主题。所有关键过程域在 P-CMM 模型上可映射为 4 个主题，它们是发展能力、团队与文化建设、激励和管理业绩、形成劳动力。参看表 10.10。

表 10.10 主题及过程分类

主题 / 成熟度级别	发展能力	团队和文化建设	激励和管理业绩	形成劳动力
5. 优化级	训练、个人业务能力发展	持续的劳动力创新		
4. 已管理级	业务指导	团队建设	企业级的业务整合、基于团队的条例	企业级的业务能力管理
3. 已定义级	业务能力、知识与技能分析	参与与分享的文化	基于业绩能力的条例、职业发展	劳动力计划
2. 可重复级	培训、交流与沟通	交流与沟通	薪酬、业绩管理、工作环境	人员组织
1. 初始级				

在进行基于 P-CMM 的改进工作时，不应把它看作仅仅是人力资源部门的事情，尤其是和现行的软件过程改进项目一起实施时更是如此。SEI 的 P-CMM 顾问委员会建议：在软件工程小组内加入人力资源管理人员，以进行基于 P-CMM 的改进工作。这样，带给软件机构管理层的信息就是：一个致力于改善整个软件过程能力的项目强调的是过程、技术和人员，缺一不可。基于 P-CMM 的改进工作和其它任何发展项目一样，第一要有计划，第二要跟踪其进展，第三要有专人负责。只有这样，才能保证 P-CMM 在企业的劳动力能力的发展方面发挥最大效益。

9. 在软件开发机构中贯彻 ISO 9000 国际标准

近年来，国际上影响最大的质量管理标准是国际标准化组织于 1987 年公布的 ISO 9000 系列标准。这一国际标准发源于欧洲经济共同体，但很快就波及美国。日本及世界各国。到目前为止，已有 100 多个国家在它们的企业中采用和实施这一系列标准。中国也采取了积极态度，确定对其等同采用，发布了与之相应的质量管理国家标准系列 GB / T 19000；同时积极组织实施和开展质量认证工作。

(1) 质量管理

无论任何产业，其产品的质量如何都是生产者、消费者，以及中间商十分关注的问题。市场的竞争很大程度上反映了在质量方面的竞争，生产者(也称供方)为使

自己提供给客户的产品达到并保持一定的质量水平，必须进行质量管理。质量管理包括以下的质量活动：

§ 确定自己的质量方针和质量目标；

§ 确定各个岗位的职责和权限；

§ 建立质量体系，并使其有效运行。

质量体系是为实施质量管理所需的组织机构、程序、过程和资源。它包括：质量策划、质量控制、质量保证及质量改进等一系列活动。 质量管理的主要类型分为 3 种：

§ 质量检验型：即对产成品进行检验。这是一种事后的、被动的方法。

§ 全面质量管理型：即从产品质量形成的全过程和企业全员努力两方面来提高工作质量以保证产品质量。这是一种积极的、主动的方法，但仅属于供方的活动。

§ 质量认证型：这是在全面质量管理基础上形成的质量管理手段，是从客户需要出发，以“确保顾客满意”为宗旨的消费者(也称需方)主导型的质量管理。目前在国际贸易中，往往是以对方是否通过了质量认证作为判断其产品质量水平和结成贸易伙伴的条件。

(2) 质量认证与质量审核 质量认证又称为合格认证。1986 年国际标准化组织(ISO)定义质量认证为：由可以充分信任的第三方证实某一经鉴定的产品或服务符合特定标准或其它技术规范的活动。

质量认证的特点：

§ 质量认证的对象是产品或服务，主要针对的是产品。其中对产品的质量认证分为对产品本身的质量认证和对产品生产过程的质量认证。

§ 认证的依据是标准。标准常常是被公认的或具有权威性的有关质量管理的国际标准或国家标准，也可能是行业标准或其它标准。认证审核就是检查对特定标准规定要求的符合性。符合标准要求就可通过审核，获得认证。

§ 通过审核、取得认证的标志是给予合格证书或准许使用合格标志。

§ 质量认证是第三方开展的活动。由具有专门技术、设备和人员的第三方，作为专门从事质量认证的机构，可以由国家权威机构授权，进行公正的、科学的质量审核和认证，因而是可以信赖的。供方和需方出具合格证明都不属于认证。

(3) ISO 9000 系列标准的特点

ISO 9000 系列标准的特点如表 10.9 所示。

表 10.9 ISO 9000 系列标准的特点

表 10.9 ISO 9000 系列标准的特点

特点	主要表现
国际性	国际组织制定，并向各国推荐，被 100 多个国家采纳为国家标准；多边互认。
完整性	从术语、质量保证、质量管理到支持性技术标准及实施指南，共计 20 个，形成完整体系。包括质量保证和管理的要求，质量体系建立，标准选用，审核人员条件，审核过程等。
兼容性	作为 ISO 9000 系列标准核心的 3 个质量保证标准不是互相孤立的，是互相包容的。ISO 9001 标准最全面，ISO 9002 和 ISO 9003 标准的适用范围依次缩小，如图 10.20 所示。
主动性	选用和实施 ISO 9000 标准，建立质量体系，出于自身提高机构内部质量管理水平的需要，叫做“管理者推动”。若出于客户要求，进而转变为供方意向，叫做“受益者推动”。无论哪一种情况向认证机构提出认证申请都是主动行为。
可信性	按 ISO 9000 标准建立的质量体系认证制度要求授权的认证机构对供方的质量体系进行公正全面、系统、公正的审核，且是独立进行。具有相当的说服力和可信性。
指导性	通常以标准为指导原则，结合机构自身的实际情况，特别是结合自身的企业文化，进一步发挥和创造，建立对自己有效的质量体系。
科学性	ISO 9000 标准的背后隐藏着现代质量管理的科学原理。一些深刻的道理只有通过实践才能有所体会。
实践性	标准的文本不是空洞的条文，它与实际工作是密切结合的，可借以指导具体的实施和操作。此外，标准基于大量的质量管理实践。向前追溯，可知它源于英国国家标准和美国军用标准。认证制度则可追溯到 20 世纪初的英国质量管理机构。

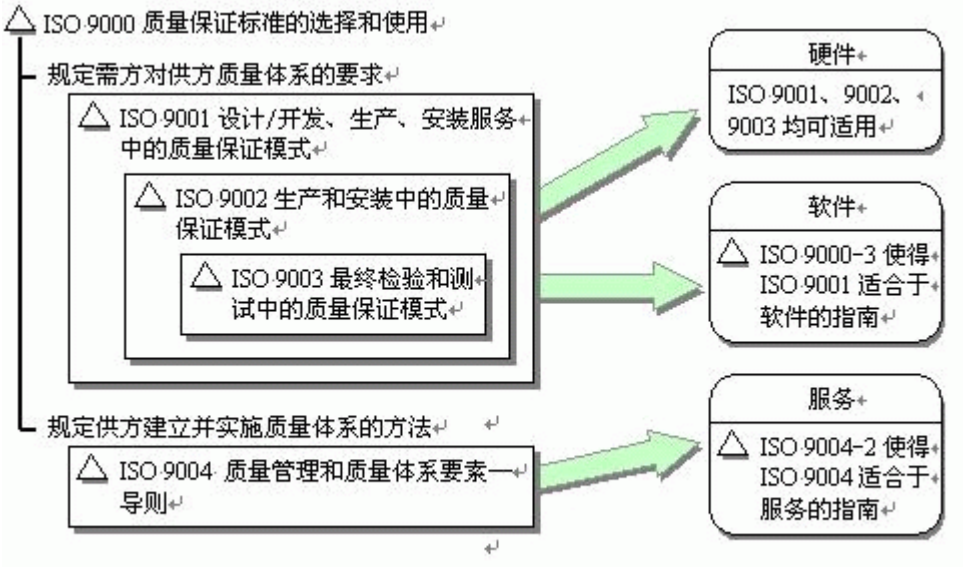


图 10.20 ISO 9000 质量保证系列标准之间的关系

(4) ISO 9000 系列标准的科学依据

ISO 9000 系列标准所体现的现代质量管理有其丰富的内涵，主要表现在以下几个方面：

① 产品质量并非在产品检验中得到，而是形成于生产的全过程。ISO 9000 叙述了需方和供方应如何进行有组织的质量保证活动，才能得到较为满意的软件；规

定了从双方签订开发合同到设计、实现以至维护整个生存期中应当实施的质量保证活动。

② 为把握产品的质量，必须使影响产品质量的全部因素在生产全过程中始终处于受控状态。为使产品达到质量要求，ISO 9000 要求开发机构建立质量体系。首先要求明确供需双方的职责，针对所有可能影响质量的各个因素都要采取有力措施，作出如何加强管理和控制的决定。对与质量有关的人员规定其职责和职权，使产品质量真正得到控制。

③ 应使企业具有持续提供符合要求产品的能力。这正是质量保证的观念。在生产机构中建立质量体系，对所有影响产品质量的因素，包括技术、管理和人员，都进行有效的控制，可减少、消除和预防质量缺陷。如果能够向管理者和顾客表明，自己的质量体系具有持续稳定地满足质量要求的能力，就能取得两方面的信任。

④ 质量管理必须坚持进行质量改进。产品质量取决于相关客户的满意程度。而这种满意程度与过程的合理组织和过程的效率直接相关。质量改进是一种以追求更高的过程效益和效率为目标的持续活动。质量改进是长期的任务。不断研究和改善质量管理制度是各级生产者的责任。

⑤ 质量管理 PDCA 循环。任何管理工作，包括质量管理，都必须按照 PDCA 循环进行。该循环体现了管理的规律，参见图 10.21。这 4 个方面缺一不可。当完成了 4 阶段的循环后，轮子已经前进，整个工作自然提升到一个新的阶段。

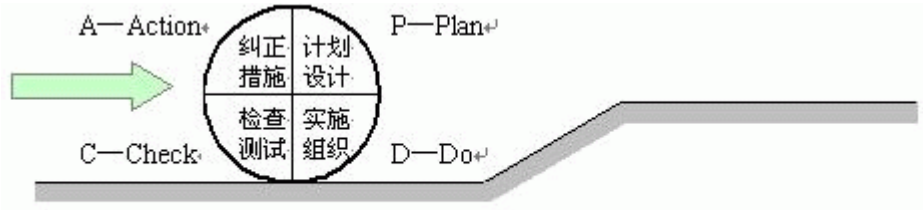


图 10.21 PDCA 循环

⑥ 质量管理的核心是预防而不是补救。成熟的生产机构的良好素质表现在建立了有效的质量保证体系以后，可以及时纠正，特别是能够预防质量问题的发生。预防是主动的，是在质量事故之前；检验和补救是被动的，事后的，也必然是代价昂贵的。

(5) ISO 9000 系列质量标准概要

ISO 9000 系列质量标准现有 20 个，可分为 5 类，下面简要做一说明。

① 质量术语标准

• ISO 8402－1994 质量管理和质量保证——术语

② 质量保证标准

• ISO 9001 质量体系——设计 / 开发、生产、安装和服务中的质量保证模式；

• ISO 9002 质量体系——生产和安装中的质量保证模式；

• ISO 9003 质量体系——最终检验和测试中的质量保证模式；

• ISO 9004 质量管理和质量体系要素——导则。

③ 质量管理标准

• ISO 9004-1 质量管理和质量体系要素——第 1 部分：指南；－通用指南

• ISO 9004-2 质量管理和质量体系要素——第 2 部分：服务指南；－针对服务业

• ISO 9004-3 质量管理和质量体系要素——第 3 部分：流程性材料指南；－针对流体、气体、粒状等特定形态产品

• ISO 9004-4 质量管理和质量体系要素——第 4 部分：质量改进指南；

④ 质量管理和质量保证标准的选用和实施指南

• ISO 9000-1 质量管理和质量保证标准——第 1 部分：选择和使用指南；

• ISO 9000-2 质量管理和质量保证标准——第 2 部分：ISO 9001、ISO 9002 和 ISO 9003 的实施通用指南；

• ISO 9000-3 质量管理和质量保证标准——第 3 部分：ISO 9001：1994 在计算机软件开发、供应、安装和维护中的使用指南；

• ISO 9000-4 质量管理和质量保证标准——第 4 部分：可信性大纲管理指南；－可信性包括可靠性、可维修性和可用性。

⑤ 支持性技术标准

• ISO 10005 质量计划指南

• ISO 10007 技术状态管理指南

• ISO 10011-1 质量体系审核指南——第 1 部分：审核；

• ISO 10011-2 质量体系审核指南——第 2 部分：质量体系审核员的评定准则；

• ISO 10011-3 质量体系审核指南——第 3 部分：审核工作管理；

• ISO 10012-1 测量设备的质量保证要求——第 1 部分：测量设备的计量确认体系；

• ISO 10012-2 测量设备的质量保证要求——第 2 部分：测量过程的控制；

• ISO 10013 质量手册编制指南。

从质量保证的要求来讲，ISO 9001 是 ISO 9000 系列标准中一个很重要的质量保证标准，也是软件机构推行质量认证工作的一个基础标准。它于 1994 年由国际标准化组织公布，我国已及时将它转化为国家推荐标准，编号为 GB / T 19001－1。

ISO 9001 标准在 20 个方面规定了供方在全部生产活动过程中的质量要求，有时人们将这 20 个方面称为 20 个质量体系要素。如表 10.10 所示。

编号	质量要素	主 要 职 责
1	管理职责	制定质量方针；对所有与质量相关的人员，包括执行人员和验证人员，规定职责、权限和相互关系；负责定期组织机构内的管理评审。
2	质量体系	建立质量体系；编制质量手册及一系列程序文件，并加以贯彻实施。
3	合同评审	在合同签订之前，对合同、标书或订单进行全面评审。使合同评审形成制度。
4	设计控制	供方在产品设计方面建立质量控制，保持稳定程序，并形成文件。其中包括：设计和开发的策划，组织上的接口和技术上的接口，确定设计输入，确定设计输出，对设计结果进行正式评审，实施设计验证和设计确认。
5	文件控制	对有关的所有文件和资料进行控制，包括审批适用性，防止使用失效或作废文件及审批更改，保证文件和资料适用、系统、协调和完整。
6	采购	供方对采购活动建立程序并加以保持。采购的质量控制包括：对分承包方的评价，对采购文件的要求和对采购产品的检验。
7	顾客提供产品的控制	供方应对顾客提供的产品建立及保持贮存和维护的控制程序并形成文件。
8	产品标识及可追溯性	供方应在接收和生产、交付及安装的各阶段以适当方式对产品进行标识，建立程序并形成文件。这种标识应具有唯一性和可追溯性。
9	过程控制	供方对直接影响质量的生产、安装和服务过程进行控制，制定程序并形成文件。
10	检验和试验	为使产品满足规定要求，供方应建立和保持检验和试验活动的程序并形成文件。包括进货、过程、最终的检验和试验以及对检验和试验的记录的要求。
11	检验、测量和试验设备	供方应对用以证实产品符合规定要求的检验、测量和试验设备建立并保持控制、校准和维修的程序，并形成文件。
12	检验和试验状态	对在检验和试验过程中处于不同状态(未检或待检、已检合格、已检不合格)的产品进行标识，防止不合格材料、半成品、部件和成品混入或误认。
13	不合格品的控制	供方应建立和保持对不合格品的控制程序并形成文件。，包括对不合格品的标识、记录、评审、隔离和处置，防止误用或安装。
14	纠正及预防措施	为消除实际已出现的不合格品及可能出现不合格品的根源，供方应建立并保持纠正和预防措施的程序并形成文件。
15	搬运、贮存、包装、防护及交付	供方应建立搬运、贮存、包装、防护及交付的程序并形成文件。
16	质量记录	供方应建立及保持对质量记录的标识、收集、编目、查阅、归档、储存、保管和处理的程序并形成文件。
17	内部质量审核	为验证质量活动和有关结果是否符合计划安排，确定质量体系的有效性，供方应对内部质量审核工作建立和保持程序并形成文件。
18	培训	对所有从事与质量有关工作的人员进行培训，要先明确培训要求并建立程序。
19	服务	在规定有服务要求的情况下，供方应建立和保持有关服务的实施、验证和报告的程序并形成文件。使服务能够满足规定要求。
20	统计技术	供方应建立并保持为分析过程能力和产品特性所采用的若干统计技术的实施程序并形成文件。

总而言之，上述各质量体系要素几乎都要求建立和保持程序并形成文件，此外还要求在执行程序的过程中保存记录。

为便于理解，将这 20 个质量体系要素分为 5 类，分别隶属于机构、管理者、工

程、支持和顾客的类别下，如图 10.22 所示。图中括号内的数字表示质量体系要素的顺序号。

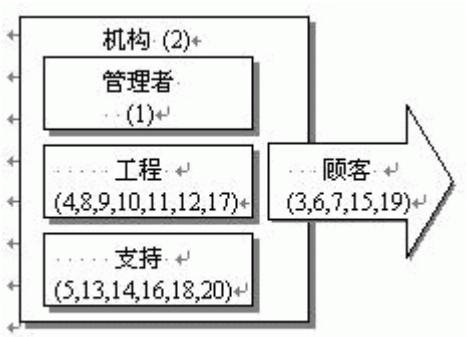


图 10.22 质量体系要素的分类

(6) ISO 9000-3 标准

ISO 9000 系列标准原是为制造硬件产品而制定的标准，不能直接用于软件制作。后来曾试图将 ISO 9001 改写用于软件开发方面，但效果不佳。于是，以 ISO 9000 系列标准的追加形式，另行制定出 ISO 9000-3 标准。这样，ISO 9000-3 就成了用于“使 ISO 9001 适用于软件开发、供应及维护”的“指南”。不过，在 ISO 9000-3 的审议过程中，日本等国曾先后提出过不少意见。所以，在内容上与 ISO 9001 已有相当不同。此外，1997 年公布的 ISO 9000-3 版本是又经过修改的版本，与我国的国家标准 GB / T19000.3-94 相比，其内容作了很大更改。

ISO 9000-3 是一个指南性的标准，其指南性表现在：

§ 从软件的角度对 ISO 9001 给出具体的说明和解释。在 ISO 9001 的 20 个质量体系要素的每一个要素后面，逐一对照作出说明和解释。

§ 指南性标准不是认证审核的依据，认证审核的依据是 ISO 9001 标准。就是说，在认证审核时，考察申请认证的软件机构针对 ISO 9001 的每一个质量体系要素的贯彻实施情况，审查有无与条文规定要求不符的情况，而不是考察有无违背 ISO 9000-3 的情况。

ISO 9000-3 实际上是一个建议性的指南，而不是强制性的。它可以理解为：可以按此解释或建议去做，但如果有更好的做法，只要能够满足相关的 ISO 9001 标准规定的要求，也可以不按它的建议去做。ISO 9000-3 对 20 个质量体系要素的解释如表 10.11 所示。

编号	质量要素	解 释
1	管理职责	涉及质量方针、组织、管理评审。对软件无特殊要求。
2	质量体系	对质量策划提出 6 条建议，如用可测量的用语表示质量要求，指明适用的生存期模型，规定阶段的启动和结束准则，明确要执行的评审，测试等活动的类型，对配置管理进行详细的策划等。对质量计划做了较为详细的说明。
3	合同评审	对软件标书、合同或订单的评审提出几点建议。包括与顾客有关的事项，技术事项，管理事项，法律、安全和保密事项等。在合同修订与记录方面无特殊要求。
4	设计控制	软件的设计控制相当于软件开发，是软件质量保证的重点，说明有以下几点。 ① 总则：阐明软件开发中主要工作的质量控制。包括指明软件开发应涉及开发策划、需求分析、体系结构设计、详细设计和程序编码等活动；软件开发项目可按一个或多个生存期模型组织，并按模型计划和实施过程、活动和任务，至于采用何种模型不限；建立和保持软件开发形成文件的程序；建议考虑设计活动的若干方面；对编码建立形成文件的规则；工具、设施和技术可用于管理，也可用于开发和服务。 ② 设计和开发的策划：阐明软件开发计划或软件项目计划制定的有关问题。包括产品开发策划的活动；开发计划的评审与批准、提交；开发策划的内容等。 ③ 组织与技术接口：规定软件开发过程中有关方面在组织和技术上的关系。包括供方与分承包方之间确定职责范围、技术信息传递方式、分承包方开发计划的评审；顾客按合同规定的职责；供方和顾客联合评审的安排；还需考虑其它参与设计、安装、维护和培训的人员。 ④ 设计输入：设计输入是软件需求规格说明，需求最好由顾客提供要求，也可由供方提供，其内容要得到顾客认可；实现规定制定需求规格说明的方法；需求的更改应加以控制；制定需求规格说明的内容；要求需求能够在验收时确认。 ⑤ 设计输出：设计输出应包括设计规格说明、源代码、用户手册等。 ⑥ 设计评审：要求制定评审计划、设计评审内容、评审记录、评审程序、最终保存评审结果。 ⑦ 设计验证：是在开发过程中对阶段输出的检验。包括设计输出评审、原型和仿真演示或测试。此外，按质量计划或程序文件制定验证计划。 ⑧ 设计确认：在产品交付顾客验收之前，确认其预期的功能和性能。 ⑨ 设计更改：规定设计的更改应加以控制。
5	文件和资料控制	① 总则：培植管理程序可用以实施文档和资料控制。包括软件文档、质量体系文件、开发管理文档等。 ② 文件和资料的批准和发布：若采用电子手段实现控制，要特别注意批准、存取、发放、媒体和归档程序。
6	采购	① 总则：采购的产品应包括市售软件、分承包方开发的软件、硬件、工具软件、合同制工作人员、维护和顾客的支持服务、培训课程和教材。 ② 对分承包方的评价：无特殊要求。 ③ 采购资料：对软件开发的采购文档应有明确的要求。 ④ 对采购产品的验证：无特殊要求。
7	顾客提供产品的控制	顾客提供的产品可能包括软件产品、开发工具、开发环境、测试数据和运行数据、硬件及顾客专有信息。
8	产品标识及可追溯性	① 任务：标识从规格说明到开发、复制与交付的所有软件项。若合同要求可适用于产品交付之后。 ② 实现方法是配置管理。包括配置管理定义；目标；提供功能；标识配置项时考虑；配置管理系统管理的产品；每一配置项应标识内容；配置管理计划内容。
9	过程控制	即对软件配置项或软件产品的复制、交付和安装的质量控制。包括复制规程；软件版本发行规程；安装和服务的考虑。
10	检验和试验	即对软件测试的质量控制。包括测试计划的内容；由第三方或顾客提供的软件产品的检验；验收前供方的确认测试；验收测试。
11	检验、测量和试验设备	即对软件测试工具的控制。包括对测试工具和测试技术的控制；对测试环境的测试。
12	检验和试验状态	即防止未经测试或测试有错的软件部件混入。应确定标识开发阶段和测试状态的方法。
13	不合格品的控制	即隔离不合格品。隔离方法：可从生产环境或测试环境转移到单独的环境；确定在哪些点控制和记录不合格品；可使用配置管理来实现；处置不合格品应注意的事项和方法。

整理人：刘海云
整理时间：2003.05

表 10.11 理解 ISO 9000-3 对 ISO 9001 的 20 个质量体系要素的解释

三、例题分析

【例 1】质量保证是为了保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动。质量保证是面向(A)的活动，是为了使产品实现(B)的功能。

软件质量保证活动即为了确定、达到和(C)需要的软件质量而进行的所有有计划、有系统的管理活动。为了提高软件的(D)和(E)，软件质量保证的主要任务主要有：正确定义用户要求；力争不重复劳动；掌握开发新软件的软件工程学方法和工具；组织外部力量协作开发；排除(F)；发挥每个开发者的能力；提高软件开发的(G)；提高(H)，即将评价、评审工作在工程实施之前就列入整个开发工程的工程计划之中。
供选择的答案：

A, B. ① 系统分析员 ② 开发者 ③ 消费者 ④ 软件需求 ⑤ 用户要求

C ~ E. ① 测试 ② 维护 ③ 质量 ④ 价格

⑤ 生产率 ⑥ 效率

F ~ H. ① 冗余 ② 无效劳动 ③ 开发方法 ④ 工程过程能力

⑤ 测试能力 ⑥ 计划和管理质量 ④ 测试和维护的效率

答案：A. ③, B. ⑤, C. ②, D. ③, E. ⑤, F. ②, G. ④, H. ⑥。其中，D、E 的答案顺序可互换。

分析：软件质量保证是为保证产品和服务充分满足消费者要求的质量而进行的有计划、有组织的活动。质量保证是面向消费者的活动，是为了使产品实现用户要求的功能，站在用户立场上来掌握产品质量的。

软件的质量保证就是向用户及社会提供满意的高质量的产品。进一步地，软件的质量保证活动也和一般的质量保证活动一样，是确保软件产品从诞生到消亡为止的所有阶段的质量的活动。即为了确定、达到和维护需要的软件质量而进行的所有有计划、有系统的管理活动。它包括质量方针的制定和展开、质量体系的建立和管理等。为了提高软件的只链个和软件开发的生产率，软件质量保证的主要任务主要有：

§ 熟练掌握正确定义用户要求的技术和支持工具；

§ 力争不重复劳动，生产和使用可复用的软件；

§ 掌握开发新软件的软件工程学方法和工具；

§ 组织外部力量协作开发的方法，建立跟踪检查的体制；

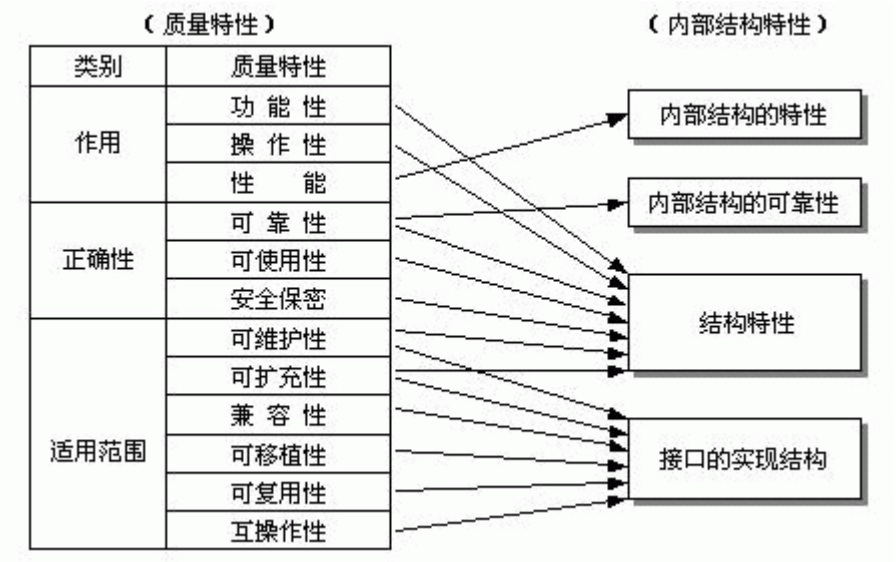
§ 排除无效劳动，力争减少因需求说明有误、设计有误而造成的返工以及重复劳动；

§ 发挥每个开发者的能力，制定技术培训计划和技术水平标准；

§ 提高软件开发的工程过程能力，即在软件开发环境的支持下，运用先进的开发技术、工具和管理方法开发软件的能力；

§ 提高计划和管理质量，将评价、评审工作在工程实施之前就列入整个开发工程的工程计划之中。

【例 2】为了实现规定的质量特性，需要把这些质量特性转换为软件的(A)的特性。软件质量需求中的“性能”，可以转换成(A)中的(B)，即每一个程序模块和(C)各自应具有的性能特性。这些性能特性的累积就形成设计规格说明中的性能特性。这种情况也适用于(D)。在质量特性中，有一些特性与功能及用户界面有关，必须把这些功能或用户界面数据正确映射到(A)中来。这时，必须对软件的(E)进行评价。此外，决定软件“适用范围”的质量特性，取决于(A)中各种(F)部分是否实现(G)。



供选择的答案：

A, B, C, E, F. ① 接口 ② 内部结构 ③ 结构特性 ④ 构成元素

⑤ 结构单元 ⑥ 性能要求 ⑦ 物理数据 ⑧ 逻辑数据

D, G. ① 模块化 ② 可靠性 ③ 适应性 ④ 性能 ⑤ 结构化

答案：A. ②, B. ④, C. ⑦, D. ②, E. ③, F. ①, G. ①。

分析：为了实现规定的质量特性，就需要把这些质量特性转换为软件的内部结构的特性。软件质量需求中的“性能”，可以转换成软件内部结构中的构成元素，即每一个程序模块和物理数据各自应具有的性能特性。这些性能特性的累积就形成设计规格说明中的性能特性。这种情况也适用于“可靠性”。

在质量特性中，有一些特性与功能及用户界面有关，必须把这些功能或用户界面数据正确映射到内部结构中来。这时，必须对软件的结构特性进行评价。此外，决定软件“适用范围”的六个质量特性，取决于软件内部结构中各种接口部分是否实现模块化。

【例 3】(A)是以提高软件质量为目的的技术活动。把(B)定义为“用户的满意程度”。为使得用户满意，有两个必要条件：(1)设计的规格说明要符合用户的要求；(2)程序要按照设计规格说明所规定的情况正确执行。把上述条件(1)称为(C)，把条件(2)称为(D)。与上述观点相对应，软件的规格说明可以分为(E)和(F)。(E)是从用户角度来看的，包括硬件 / 软件系统设计(在(G)阶段进行)、功能设计(在需求分析阶段与概要设计阶段进行)，而(F)是为了实现(E)的更详细的规格。

对(E)进行(A)时，(A)对象是在需求分析阶段产生的软件需求规格说明、数据要求规格说明，在软件概要设计阶段产生的软件概要设计规格说明等。

供选择的答案：

A, B. ① 技术创新 ② 管理评审 ③ 技术评审 ④ 过程改进

⑤ “质量” ⑥ “数量”

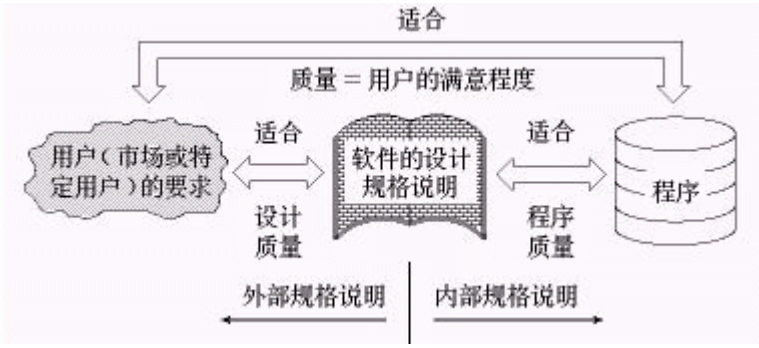
C, D. ① 程序流程 ② 程序质量 ③ 设计要求 ④ 设计质量

E ~ G. ① 内部规格说明 ② 外部规格说明

③ 概要设计 ④ 详细设计 ⑤ 系统分析 ⑥ 需求分析

答案：A. ③, B. ⑤, C. ④, D. ②, E. ②, F. ①, G. ⑤。

分析：技术评审是以提高软件质量为目的的技术活动。为此，首先要明确什么是软件的质量。缺乏质量概念的技术评审只是一种拘于形式的为评审而评审的盲目工作。通常，把“质量”定义为“用户的满意程度”。为使得用户满意，有两个必要条件：(1)设计的规格说明要符合用户的要求；(2)程序要按照设计规格说明所规定的情况正确执行。我们把上述条件(1)称为“设计质量”，把条件(2)称为“程序质量”。优秀的程序质量是构成好的软件质量的必要条件，但不是充分条件。



与上述质量的观点相对应，软件的规格说明可以分为外部规格说明和内部规格说明。外部规格说明是从用户角度来看的规格，包括硬件／软件系统设计(在系统分析阶段进行)、功能设计(在需求分析阶段与概要设计阶段进行)，而内部规格说明是为了实现外部规格的更详细的规格，即程序模块结构设计与模块处理加工设计(在概要设计与详细设计阶段进行)。因此，内部规格说明是从开发者角度来看的规格说明。将上述两个概念联系起来，则可以说设计质量是由外部规格说明决定的，程序质量是由内部规格说明决定的。

针对外部规格说明进行设计评审时，评审对象是在需求分析阶段产生的软件需求规格说明、数据要求规格说明，在软件概要设计阶段产生的软件概要设计规格说明等。

【例 4】评审软件是否有可扩充性，需要考虑可能的扩充、（ A ）和（ B ）。而软件的（ C ）是指当软件功能扩充了之后，其已有功能还能照原样使用的特性。注意（ C ）与（ D ）有区别。（ D ）是指软件运行环境改变时，可不改变软件的规格而能照原样工作的特性。（ D ）是与（ C ）相反的概念。

在评审（ C ）时，要考虑软件结构上的（ E ），即如果一个软件由多个模块构成，在改变运行环境时，仅修改或更换因环境不同而受影响的那些（ F ）就可达到（ C ）。
供选择的答案：

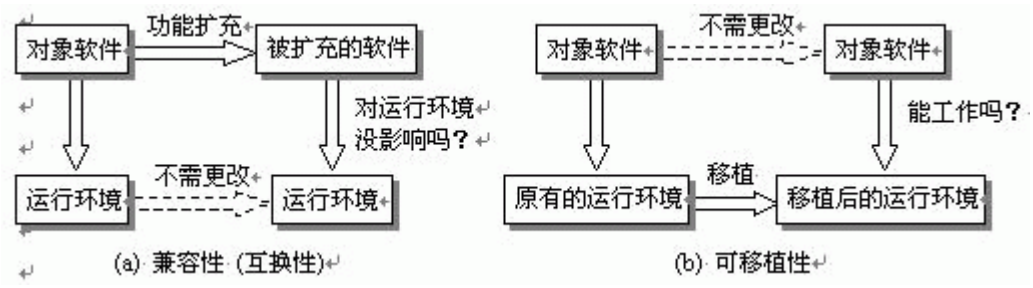
A, B. ① 结构化 ② 模块化 ③ 模块耦合性 ④ 模块通用性

C, D. ① 可移植性 ② 兼容性 ③ 可修改性 ④ 互操作性

E. ① 简明性 ② 清晰性 ③ 稳定性 ④ 结构性

F. ① 程序 ② 模块 ③ 语句 ④ 指令

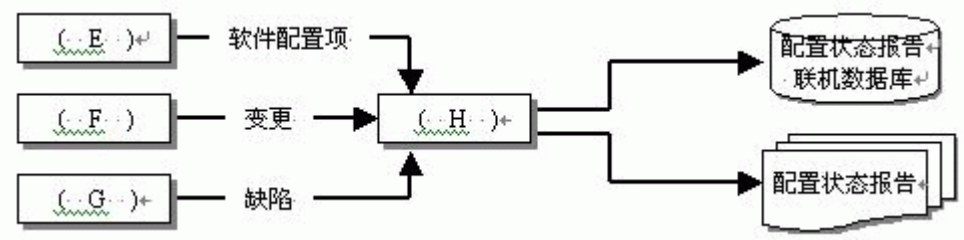
答案：A. ②, B. ④, C. ②, D. ①, E. ③, F ②。其中，A、B 的答案顺序可互换。分析：评审软件是否有可扩充性，需要考虑可能的扩充、模块化和模块的通用性。而软件的兼容性(或互换性)是指当软件功能扩充了之后，其已有功能还能照原样使用的特性。注意兼容性与可移植性有区别。，两者的比较参看图示。



可移植性是指软件运行环境改变时，可不改变软件的规格而能照原样工作的特性。可移植性是与兼容性相反的概念。兼容性表明在扩充了软件功能后，不影响(不改变)已有软件的运行环境；可移植性表明软件运行环境改变时，可不改变原有软件的规格说明而能照原样工作。

在评审兼容性时，要考虑软件结构上的稳定性，即如果一个软件由多个模块构成，在改变运行环境时，仅修改或更换因环境不同而受影响的那些模块就可达到兼容性。

【例 5】按照软件配置管理的原始指导思想，受控制的对象应是(A)。(B)将软件配置管理定义成一门管理学科；(C)将软件配置管理定义成一种标识、组织和控制修改的技术；(D)指出配置管理过程是在整个软件生存期中实施管理和技术规程的过程。软件配置状态报告见图：



供选择的答案：

A. ① 软件元素 ② 软件项目 ③ 软件配置项 ④ 软件过程

B ~ D. ①《GB / T 11457-1995 软件工程术语》

②《ISO / IEC 12207-1995 信息技术－软件生存期过程》

③《ISO 9000-3：1997 质量管理和质量保证标准 第三部分：ISO 9001：1994 在计算机软件开发、供应和维护中的使用指南》

④ 巴比奇(W.Babich)

E ~ H. ① 配置审核 ② 配置标识 ③ 配置控制 ④ 配置状态报告

⑤ 版本控制 ⑥ 基线与变更控制

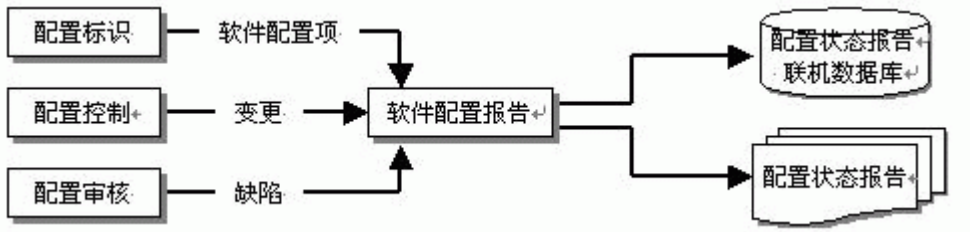
答案：A. ③, B. ③, C. ④, D. ②, E. ②, F ③, G. ①, H. ④。

分析：在软件开发的各个阶段中得到的阶段产品并非是固定不变的。设计规格说明、程序，甚至需求规格说明都可能在开发过程的某些时刻出于某种原因需要变更。软件的这种经常变更的情况必须按一定的受控方式进行，否则就要出现混乱。这就是软件配置管理的原始指导思想。受控制的对象叫做软件配置项。 关于什么是软件配置管理，已出现多种定义。下面列出几种。

《ISO／IEC 12207-1995 信息技术－软件生存期过程》：配置管理过程是在整个软件生存期中实施管理和技术规程的过程。它标识、定义系统中的软件项并指定基线；控制软件项的修改和发行；记录和报告软件项的状态和修改申请的完整性、协调性和正确性；以及控制软件项的储存、装载和交付。《ISO 9000-3：1997 质量管理和质量保证标准 第三部分：ISO 9001：1994 在计算机软件开发、供应和维护中的使用指南》：软件配置管理是一门管理学科，它对配置项的开发和支持生存期给予技术上和管理上的指导。配置管理的应用取决于项目的规模、复杂程度和风险大小。

巴比奇(W.Babich)：软件配置管理能够协调软件的开发，使得混乱减小到最小。软件配置管理是一种标识、组织和控制修改的技术，其目的是最有效地提高生产率。

《GB／T 11457-1995 软件工程术语》：软件配置管理是标识和确定系统中配置项的过程，在系统的整个生存周期内控制这些项的投放和更动，记录并报告配置的状态和更动要求，验证配置项的完整性和正确性。 配置状态报告能够清楚、及时地记载软件配置的变化。每次新分配一个软件配置项或更新一个已有软件配置项的标识，或者一项变更申请被变更控制负责人批准，并产生了一个工程变更顺序时，在配置状态报告中就要增加一条变更记录条目。一旦进行了配置审核，其结果也应当写入报告之中。配置状态报告可以放在联机数据库中，以便软件开发人员或软件维护人员可以对它进行查询或修改。此外，软件配置报告中新登录的变更应及时通知给软件管理人员和软件开发人员。



【例 6】软件过程不断改进是(A)的基本原理之一，是(B)的基本过程之一。软件过程需要不断完善，首先从(C)的软件开发方式改变为(D)的软件开发方式，按照软件过程的系统方法进行软件的工程活动和管理活动，进而不断完善软件各个软件过程，从而不断提高(E)。(E)的提高首先需要对当前的软件过程状况进行科学的(F)。

供选择的答案：

A, B. ① 软件过程 ② 软件工程七原理 ③ 软件生存期 ④ 软件需求

C, D. ① 工程化 ② 过程化 ③ 非过程化 ④ 非工程化

E. ① 软件工程能力 ② 软件过程能力 ③ 软件成熟度模型

F. ① 度量 ② 估算 ③ 评估 ④ 管理

答案：A. ②, B. ③, C. ④, D. ①, E. ②, F ③。

分析：软件过程不断改进是软件工程七原理的基本原理之一，也是软件生存期的基本过程之一。软件工程界的专家历来很重视软件过程的研究，20 世纪 70 年代形成了软件生存期的概念，1995 年正式发表了一个国际标准：ISO / IEC 12207-1995 信息技术－软件生存期过程，这是软件工程过程研究的重要成果。实践表明，软件过程需要不断完善，首先从非工程化的软件开发方式改变为工程化的软件开发方式，按照软件过程的系统方法进行软件的工程活动和管理活动，进而不断完善软件各个软件过程，从而不断提高软件过程能力。随着这种能力的提高，一个软件机构在完成一个产品时在预算、进度，特别时产品质量方面的风险就会逐步降低。显然，软件过程能力的提高首先需要对当前的软件过程状况进行科学的评估。

【例 7】CMM 提供了一个框架，将软件过程改进的进化步骤组织成 5 个成熟度等级，为过程不断改进奠定了(A)的基础。。这 5 个成熟度等级定义了一个(B)的尺度，用来衡量一个软件机构的(C)和评价其软件过程能力。每一个成熟度等级为继续改进过程提供了一个(D)。每一个等级包含了一组(E)，通过实施相应的一组(F)达到这一组(E)。5 个成熟度等级各有其不同的行为特征，通过 3 个方面来表现，即一个机构为建立或改进软件过程所进行的活动，对每个项目所进行的活动和所产生的跨越各项目的过程能力。

供选择的答案：

A, B. ① 无序 ② 有序 ③ 循环 ④ 循序渐进

C ~ F. ① 基本特征 ② 关键实践 ③ 关键过程域 ④ 台基

⑤ 过程目标 ⑥ 成熟度框架 ⑦ 软件过程成熟度

答案：A. ④, B. ②, C. ⑦, D. ④, E. ⑤, F ③。

分析：CMM 提供了一个框架，将软件过程改进的进化步骤组织成 5 个成熟度等级，为过程不断改进奠定了循序渐进的基础。。这 5 个成熟度等级定义了一个有序的尺度，用来衡量一个软件机构的软件过程成熟度和评价其软件过程能力，还能帮助软件机构自己对其改进工作排列优先次序。成熟度等级已得到确切定义，它是向成熟的软件机构前进途中的平台，每一个成熟度等级为继续改进过程提供了一个台基。每一个等级包含了一组过程目标，通过实施相应的一组关键过程域达到

这一组过程目标。当目标满足时，可使软件过程的一个重要成分稳定下来。每达到成熟度框架的一个等级，就建立起软件过程的一个相应成分，从而导致软件机构过程能力得到一定程度的提高。

5 个成熟度等级各有其不同的行为特征，通过 3 个方面来表现，即一个机构为建立或改进软件过程所进行的活动，对每个项目所进行的活动和所产生的跨越各项目的过程能力。

【例 8】P-CMM 将软件人员能力成熟度模型的本质用于软件机构的劳动力能力发展工作，它可以更容易地集成到已有的软件过程改进的项目中，指导与人员管理有关的工作。P-CMM 的价值取决于（ A ），通常不外乎两种应用方式：一是作为（ B ），评价劳动力工作的水平；二是作为（ C ），用于改进和规范劳动力工作。当 P-CMM 评估与软件过程评估联合进行时，用于 P-CMM 评估的数据应（ D ）。因为软件过程评估的分析对象是（ E ），而人员能力评估的分析对象是（ F ）。作为（C），P-CMM 的作用是为（G）提供框架，帮助企业找出劳动力条例体系中的（H）。

供选择的答案：

A. ① 软件质量 ② 软件改进过程 ③ 软件企业如何使用它 ④ 软件企业目标

B, C. ① 发展的指南 ② 评估的标准 ③ 测试的目标 ④ 维护的方法

D. ① 集中收集 ② 分散应用 ③ 统一规划 ④ 单独收集

E ~ H. ① 宏观发展 ② 微观应用 ③ 软件项目 ④ 薄弱环节

⑤ 企业的部门和部门中劳动力条例的实施情况 ⑥ 关键过程域

答案：A. ③, B. ②, C. ①, D. ④, E. ③, F ⑤, G. ①, H. ④。

分析：P-CMM 的价值取决于软件企业如何使用它，通常不外乎两种应用方式：一是作为评估的标准，评价劳动力工作的水平；二是作为发展的指南，用于改进和规范劳动力工作。当 P-CMM 评估与软件过程评估联合进行时，用于 P-CMM 评估的数据既单独收集，也可以结合企业内的其它评估一起进行，如软件过程评估、人员意向评估等。因为软件过程评估的分析对象是软件项目，而人员能力评估的分析对象是企业的部门和部门中劳动力条例的实施情况。作为企业劳动力能力发展的指南，P-CMM 并不定义具体的劳动力条例。企业应根据自己的历史、行业特点及文化背景来选择、制定自己的劳动力条例。P-CMM 的作用是为宏观发展提供框架，帮助企业找出劳动力条例体系中的薄弱环节。

【例 9】所谓进行质量管理就是在自己的机构内开展下列的质量活动：确定自己的（A）和（B）；确定各个岗位的职责和权限；建立质量体系，并使其有效运行。（ C ）是为实施质量管理所需的组织机构、程序、过程和资源。（ D ）包括产品策划、管理和作业策划，以及质量计划的编制和（ E ）的准备工作。（ E ）是以追求更高的效益和效率为目标的持续性活动；（ F ）采取某些特定作业技术或展开某

些活动以达到质量要求；（ G ）是供方为使用户确信能够满足质量要求所开展的有计划的和系统的活动，将所有影响质量的因素都得到有效的控制，从而证实确有减少、消除和预防出现质量缺陷的机制。 供选择的答案：

A ~ G. ① 质量改进 ② 质量保证 ③ 质量控制 ④ 质量体系

⑤ 质量策划 ⑥ 质量方针 ⑦ 质量维护 ⑧ 质量目标

⑨ 质量管理

答案：A. ⑥, B. ⑧, C. ④, D. ⑤, E. ①, F ③, G. ②。

其中，A、B 的答案顺序可互换

分析：所谓进行质量管理就是在自己的机构内开展下列的质量活动：

§ 确定自己的质量方针和质量目标；

§ 确定各个岗位的职责和权限；

§ 建立质量体系，并使其有效运行。

质量体系是为实施质量管理所需的组织机构、程序、过程和资源。它涉及到：

§ 质量策划：包括产品策划、管理和作业策划，以及质量计划的编制和质量改进的准备工作。

§ 质量改进是以追求更高的效益和效率为目标的持续性活动；

§ 质量控制：采取某些特定作业技术或展开某些活动以达到质量要求；

§ 质量保证：是供方为使用户确信能够满足质量要求所开展的有计划的和系统的活动，将所有影响质量的因素都得到有效的控制，从而证实确有减少、消除和预防出现质量缺陷的机制。

四、习题

【10-1】软件质量必须在设计和实现过程中加以保证。为了确保每个开发过程的质量，防止把差错传播到下一个过程，必须进行（ A ）。（ A ）是质量保证活动的一个重要部分，其目的有两个：其一是切实搞好开发阶段的管理；其二是（ B ）软件差错给用户造成损失。（ A ）的实施有两种方式：（ C ）和（ D ）。（ C ）即白盒测试和黑盒测试。（ A ）的类型有（ E ）、（ F ）、（ G ）和（ H ）。

供选择的答案：

A. ① 质量保证 ② 差错检测 ③ 质量检验 ④ 质量管理

B. ① 预先防止 ② 完全避免 ③ 减少 ④ 消除

C, D. ① 鉴定 ② 测试 ③ 实际运行管理 ④ 实际运行检验

E ~ H. ① 供货检验 ② 集中测试 ③ 中间检验/阶段评审

④ 产品检验 ⑤ 用户检测 ⑥ 程序检测 ④ 验收检验

【10-2】为了开发高质量的软件，从计划阶段开始，不但要明确软件的功能，还要明确软件应当达到什么样的质量标准，即指定软件的质量指标。软件质量度量和保证的条件通常有以下几项：（ A ），即必须制定能够适应用户要求、软件类型和规模的质量标准；（ B),即人人都容易掌握；（ C ），即对于同一软件评价的结果必须一致；；（ D ），即各阶段确立质量目标并落实；（ E ），即从不同角度加以评价；（F ）。软件的质量评价标准分为 3 级：（ G)、（ H)和(I)。（ G ）的着眼点是“是否满足用户的要求”；（ H)的着眼点是“开发者在设计实现时是否按照软件需求保证了质量”；（ I)是为定量度量软件质量而规定的一些检查项目。它们一级比一级（ J ），一级比一级易于（ K ）。

供选择的答案：

A ~ F. ① 可靠性 ② 易学习性 ③ 适应性 ④ 针对性

⑤ 客观性 ⑥ 经济性 ⑦ 主观性 ⑧ 多样性

G ~ I. ① 子质量特性 ② 质量度量准则 ③ 质量特性 ④ 质量维护准则

⑤ 质量管理准则

J, K. ① 抽象 ② 具体 ③ 检验 ④ 定量评价

【10-3】从技术上改进软件的开发过程，提高软件产品的质量无非是两个方面：一是提高（ A ），二是改进（ B ）。在发现错误和排除错误方面更重要也是更困难的是（ C ）。由于软件测试技术方面没有多少新的突破，人们只能用加强阶段评审或是检查作为辅助手段。这是一个由同行人员小组（ D)所开发的阶段产品的验证方法。至于改进(B)的新技术，是采用面向对象的开发技术或是建立(E)。一个诱人的说法是采用（ F)技术，其基本思想在于（ G)开发过程，使得差错或缺陷不可能有机可乘混入开发过程。

供选择的答案：

A, B. ① 测试效率 ② 开发速度 ③ 开发过程 ④ 维护过程

⑤ 测试方法 ⑥ 开发工具

C. ① 排除错误 ② 发现错误

D. ① 机器检查 ② 人工检查 ③ 集成测试 ④ 单元测试

E, F. ① 智能 ② 软件原型 ③ “净室”软件开发 ④ 基于构件的复用

G. ① 简化 ② 结构化 ③ 净化 ④ 强化

【10-4】软件配置项是软件配置管理的对象，指的是软件工程过程中所产生的（ A ）。不仅如此，而且在不同的时期，出于不同的要求，可进行各种组合，如针对不同的硬件环境和（ B ）的组合，这就是软件配置的概念。实施软件配置管理要做的事情有：制定（ C ）；实施（ D ）；实施版本管理和发行管理。制定适当的命名规则是（ E ）的重要工作，命名要求：（ F ），目的在于避免出现重名，以免造成混乱；（ G ），以反映命名对象之间的关系。

供选择的答案：

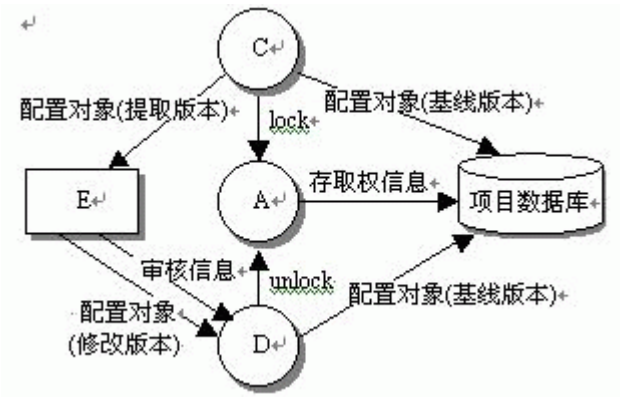
A, B. ① 接口 ② 软件环境 ③ 信息项 ④ “版本”

C ~ E. ① 配置标识 ② 配置管理 ③ 配置管理计划 ④ 变更管理

⑤ 版本变化 ⑥ 配置审核

F, G. ① 多态性 ② 唯一性 ③ 独立性 ④ 可追溯性

【10-5】在变更管理中，“检出”和“登录”处理实现了两个重要的变更控制要素，即（ A ）和（ B ）。（ A ）控制和管理各个软件技术人员存取或修改一个特定软件配置对象的权限；（ B ）可用来确保由不同的人所执行的并发的变更不会产生混乱。请选择适合的答案完成右图所示的存取和控制图。



供选择的答案：

A, B. ① 异步控制 ② 同步控制

③ 存取控制 ④ 基线控制

C, D. ① 登入 ② 检出 ③ 管理 ④ 填写变更请求

E. ① 审查人员 ② 配置人员 ③ 质量保证人员 ④ 软件工程人员

【10-6】软件工程标准的类型是多方面的。它可能包括(A)标准，如方法、技术、度量等；(B)标准，如需求、设计、部件、描述、计划、报告等；(C)标准，如职别、道德准则、认证、特许、课程等；(D)标准，如术语、表示法、语言等。根据中国国家标准 GB / T 15538-1995(软件工程标准分类法)规定，软件工程标准可用一张(E)来表示。

供选择的答案：

A ~ D. ① 专业 ② 产品 ③ 记法 ④ 过程 ⑤ 管理

E. ① 矩阵图 ② 一维的表格 ③ 二维的表格 ④ 帕列特图

【10-7】根据软件工程标准制定的机构和标准的适用范围有所不同，它可分为五个级别，即(A)、(B)、(C)、(D)和(E)。(A)由国际联合机构制定和公布，通常冠有 ISO 字样；而 GB、ANSI、JIS 等属于(B)；IEEE、GJB 等属于(C)；(D)的实例是(F)；(E)是专用的软件工程规范，如(G)。

供选择的答案：

A ~ E. ① 国家标准 ② 国际标准 ③ 企业规范 ④ 项目规范

⑤ 行业标准

F, G. ① DOD-STD ② FIPS(NBS) ③ BS ④ MIL-S

⑤ IBM 的《程序设计开发指南》 ⑥ CIMS 的《软件工程规范》

【10-8】下面关于标准和文档的叙述中正确的叙述为(、 、 、 、)。

① 国家标准是由政府或国家级机构制定或批准，适用于全国的标准。这些标准都是强制性的，相关产品必须严格执行标准。 ② ISO9001 是设计 / 开发、生产、安装和服务中的质量保证模式，ISO9000-3 是使 ISO9001 适合于软件的质量保证指南。

③ 软件工程标准化可提高软件的生产率。

④ 软件质量保证体系是贯穿于整个软件生存期集成化过程体系，而不仅仅体现在最后产品的检验上。

⑤ ISO9000-3 与具体的开发模式有关。它将软件全过程工序从管理角度、合同角度和工程角度划分为三大类。

⑥ 软件测试计划始于需求分析阶段，完成于软件设计阶段。

⑦ 任何一个文档都应是完整的、独立的，它应自成体系。

⑧ 在新文档取代旧文档后，管理人员应删去旧文档。

⑨ 软件开发机构应保存一份完整的主文档，并允许开发人员可以保存主文档中的部分主文档，有自己的活动空间。

⑩ 软件需求分析报告是给开发人员使用的，不是给其它人员，如维护人员，用户等使用的。

【10-9】从供选择的答案中选出同下列各条叙述关系最密切的字句。

(1) 在软件开发中以下几方面的内容应分别在哪个文档中得到阐明：

A) 软件总体结构 B) 运行环境 C) 出错处理设计

(2) 以下两个文档应分别在哪两个阶段中开发：

D) 初步的用户手册 E) 确认测试计划

供选择的答案：

A ~ C. ① 可行性研究报告 ② 项目开发计划 ③ 软件需求规格说明

④ 数据要求规格说明 ⑤ 概要设计规格说明 ⑥ 详细设计规格说明

⑦ 测试计划 ⑧ 测试报告 ⑨ 用户手册

D, E：① 可行性研究与计划 ② 需求分析 ③ 概要设计

④ 详细设计 ⑤ 测试 ⑥ 维护

【10-10】从下列关于文档编制的叙述中选出五条正确的叙述。

① 可行性研究报告应评述为了合理地达到开发目标而可能选择的各种方案，以便用户抉择。因此，编写者不必提出结论。

② 操作手册的编写工作应该在软件测试阶段之前完成。

③ 软件的开发单位应该建立本单位文档的标识方法，使文档的每一页都具有明确的标识。

④ 为了使得文档便于修改保持一致性，各文档的内容不应有相互重复的地方。

⑤ 用户手册要使用专门术语，并充分地描述该软件系统的结构及使用方法。

⑥ 详细设计说明书中可以使用判定表及必要的说明来表示程序的逻辑。

⑦ 概要设计说明书中可以使用 IPO 图来说明接口设计。

⑧ 测试分析报告应把每个模块实际测试的结果，与软件需求规格说明书和概要设计说明书中规定的要求进行对照并作出结论。

⑨ 软件需求规格说明书中可以对软件的操作人员和维护人员的教育水平和技术专长提出要求。

⑩ 项目开发计划除去规定项目开发所需的资源、开发的进度等以外，还可以包括用户培训计划。

【10-11】 保证软件过程质量应包含的工作有：(A)过程标准，(B)开发过程，(C)软件过程。近年来，软件工程界的专家对过程评估和过程改进表现出浓厚的兴趣。其中有卡内基-梅隆大学 SEI 的(D)；ISO 的(E)。(D)的主题是(F)，(E)的主题是(G)。Bell 的(I)的主题是(H)。

供选择的答案：

A ~ C. ① 监控 ② 报告 ③ 定义 ④ 执行

D, E, I. ① ISO 9001, ISO 9000-3 ② CMM

③ ISO/IEC TR15504 SPICE ④ MIL-STD-498

⑤ TRILLTUM ⑥ V-Model ⑦ BooTSTRAP

F ~ H. ① 软件过程改进和能力评估 ② 建立和维持质量体系

③ 软件过程能力评估 ④ 软件过程评估（基于 CMM）

⑤ 软件过程定义

【10-12】人们用于开发和维护软件及其相关产品的一系列活动称为(A)；表示(开发组织或项目组)遵循其软件过程所获得的实际结果称为(B)；而描述通过遵循其软件过程能够实现预期结果的程度称为(C)；一个特定软件过程被明确和有效地定义、管理、测量和控制的程度称为(D)；表征(D)的平台称为(E)；对软件机构进化阶段的描述，随着软件机构定义、实践、测量、控制和改进其软件过程，软件机构的能力经过这些阶段逐步前进，称为(F)；互为关联的若干软件实践活动和有关基础设施的一个集合称为(G)；对(G)的实施起关键作用的方针、规程、措施、活动以及相关基础设施的建立称为(H)。

供选择的答案：

A ~ H. ① 关键过程域 ② 软件能力成熟度模型 ③ 关键实践

④ 软件过程 ⑤ 软件过程能力 ⑥ 软件过程成熟度

⑦ 软件过程性能 ⑧ 软件能力成熟度等级

【10-13】基于软件人员能力成熟度模型的评估的依据是成熟度级别上的(A)。一个企业的成熟度级别是所有(A)被实施的(B)级别。软件企业通过评估结果可以了解(A)方面的强项和弱项，明确努力的方向。P-CMM 模型并不禁止处于较低成熟度级别上的企业在需要的时候实施高一级关键过程域的内容。但是，由于每一级都是向上一级进化的基础，所以，跳越成熟度级别的进化是(C)。基于 P-CMM 的改进工作和其它任何发展项目一样，第一要有(D)，第二要(E)，第三要有专人负责。卡内基－梅隆大学 SEI 的 P-CMM 顾问委员会建议：在软件工程小组内加入人力资源管理人员，以进行基于 P-CMM 的改进工作。这样，带给软件机构管理层的信息就是：一个致力于改善整个软件过程能力的项目强调的是(F)，缺一不可。

供选择的答案：

A, B. ① 最高 ② 最低 ③ 中间 ④ 关键过程域

⑤ 实践 ⑥ 软件过程改进

C. ① 无风险的 ② 有风险的 ③ 盲目的 ④ 可预测的

D, E. ① 方法 ② 计划 ③ 跟踪其发展 ④ 加速其进程

F. ① 过程、技术和人员 ② 人力、软件和硬件 ③ 目标、方法和过程

④ 计划、测试和维护 ⑤ 过程、评估和检验

【10-14】ISO 9000 族标准是指国际标准化组织中的质量管理和质量保证技术委员会(ISO / TC176)制定的标准，现有(A)个标准，可分为 5 类：质量术语标准，如(B)。(C),如 ISO 9001、ISO 9002、ISO 9003 系列标准。(D)，如 ISO 9004 系列标准。(E)，如 ISO 9000 系列标准。(F)，如 ISO 10005 质量计划指南，ISO 10007 技术状态管理指南等。

供选择的答案：

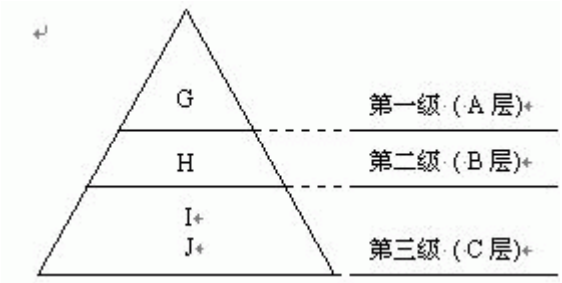
A. ① 10 ② 25 ③ 20 ④ 30

B. ① ISO 9000-3 ② ISO 9004-4 ③ ISO 9001 ④ ISO 8402

C ~ F. ① 支持性技术标准 ② 质量管理和质量保证标准的选用和实施指南

③ 质量保证标准 ④ 质量管理标准 ⑤ 质量改进标准

【10-15】质量体系是一种(A)。质量体系是通过若干(B)来实现的。(B)是将(C)转化为(D)的一组彼此相关的资源和活动。(B)本身具有(E)的效果，是一种有效益的行为。(F)是通过质量体系来实现的。一个组织机构建立自己的质量体系，并使之有效地运行，使达到(F)目标的重要手段。请选择合适的答案，完成右图的质量体系文件。



供选择的答案：

A. ① 体系结构 ② 管理手段 ③ 质量改进过程 ④ 质量管理制度

B ~ E. ① 输出 ② 输入 ③ 接口 ④ 减少

⑤ 增值 ⑥ 过程 ⑦ 扩充

F. ① 质量保证 ② 质量管理 ③ 质量检验 ④ 质量改进

G ~ J. ① 质量记录 ② 程序文件 ③ 质量手册 ④ 作业指导书

【10-16】(A)是计算机软件机构实施 ISO 9001 的指南性标准。它的指南性主要表现在：ⅰ）对于针对的标准给予特定的(B)，ISO 9001 提供了(C)个质量体系要素。ⅱ）指南性的标准不是(D)的依据。在 ISO 9001 的质量体系要素的每一条都强调了要求的强制性，而在(A)中是建议性指南。由于 ISO 9001 标准本来是针对传统的制造业制定的，而软件业又有许多不同于制造业的特性，所以，(A)起了桥梁作用。

此外，(E)将整个软件生存期分为 17 个过程，并且对每一个过程按(F)的三个层次具体做了解释，为我们进一步理解(A)提供了帮助。

供选择的答案：

A, E. ① ISO 9001 ② ISO 9002 ③ ISO 9000-3 ④ ISO／IEC 9126

⑤ ISO 8402 ⑥ ISO／IEC 12207

B, D. ① 说明与解释 ② 强制性 ③ 建议性 ④ 认证审核

C. ① 10 ② 20 ③ 25 ④ 30

F. ① “任务-活动-过程” ② “活动-任务-过程”

③ “过程-活动-任务” ④ “任务-过程-活动”

五、习题解答

【10-1】A. ③, B. ①, C. ④, D. ①, E. ①, F. ③, G. ④, H. ④。其中，E、F、G、H 的答案顺序可互换。

软件质量必须在设计和实现过程中加以保证。如果工程过程能力不够，或由于各种失误产生软件差错，其结果就会产生软件失效。为了确保每个开发过程的质量，防止把差错传播到下一个过程，必须进行质量检验。检验目的有两个：其一是切实搞好开发阶段的管理，检查各开发阶段的质量保证活动开展得如何；其二是预先防止软件差错给用户造成损失。质量检验是质量保证活动的一个重要部分。

检验的实施有两种方式：实际运行检验(白盒测试和黑盒测试)和鉴定。检验的类型有供货检验(指委托外单位承担的开发作业，而后买进或转让的产品)、中间检验／阶段评审、产品检验和验收检验(确认是否已达到进行“产品检验”的质量要求)。

【10-2】A. ③, B. ②, C. ①, D. ④, E. ⑤, F. ⑥, G. ③, H. ①, I. ②, J. ②,

K. ④。

为了开发高质量的软件，从计划阶段开始，不但要明确软件的功能，还要明确软件应当达到什么样的质量标准，即指定软件的质量指标。

为了达到这些质量指标，在开发过程的各个阶段需要进行检查和评价。软件质量度量和保证的条件通常有以下几项： § 适应性：即必须制定能够适应用户要求、软件类型和规模的质量标准；

§ 易学习性：即不需要特殊技术，软件技术人员人人都容易掌握；

§ 可靠性：对于同一软件，尽管评价的人和场合不同，但结果必须一致；

§ 针对性：即各阶段应确立质量目标并实施落实；

§ 客观性：即应从不同角度加以评价；

§ 经济性：即应考虑如何才能把质量度量和保证所需要的费用控制在适当的范围内。

软件的质量评价标准分为 3 级：质量特性、子质量特性和质量度量准则。质量特性的着眼点是“是否满足用户的要求”；子质量特性的着眼点是“开发者在设计

实现时是否按照软件需求保证了质量”；质量度量准则是为定量度量软件质量而规定的一些检查项目。它们一级比一级具体，一级比一级易于定量评价。

【10-3】A. ①, B. ③, C. ②, D. ②, E. ②, F. ③, G. ③。

从技术上改进软件的开发过程，提高软件产品的质量无非是两个方面：一是提高测试工作的效率，更有效地发现和排除软件开发过程中发生的各种差错；二是改进开发过程，使得各种差错不被引进或更少地被引进，从而减轻排错的工作量。

在发现错误和排除错误方面更重要也是更困难的是发现错误，也就是测试。因为只有发现了错误才能排除它。由于近年来软件测试技术方面没有多少新的突破，人们只能用加强阶段评审或是检查作为辅助手段。这是一个由同行人员小组人工检查所开发的阶段产品的验证方法，其目的在于发现阶段产品中的缺陷，继而加以排除。至于改进开发过程的新技术，可以采用面向对象的开发技术或是建立软件原型，多少会收到一些效果，但无论如何不能说从根本上解决了软件的质量问题。一个诱人的说法是采用“净室”软件开发技术，其基本思想在于净化软件开发过程，使得差错或缺陷不可能有机可乘混入开发过程。

【10-4】A. ③, B. ②, C. ③, D. ④, E. ①, F. ②, G. ④。

软件配置项是软件配置管理的对象，指的是软件工程过程中产生的所有信息项。包括计算机可执行的源代码、目标码、数据库等，以及计算机不可执行的文档、源程序清单、测试用例等。随着软件工程项目的进展，软件配置项会逐渐增多，于是配置管理的作用就会充分地按照显示出来。不仅如此，而且在不同的时期，出于不同的要求，可进行各种组合，如针对不同的硬件环境和软件环境的组合，这就是软件配置的概念。

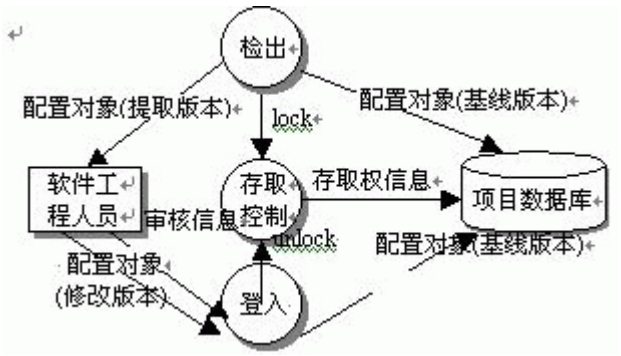
实施软件配置管理要做的事情至少有：

§ 制定配置管理计划。在软件工程项目制定开发计划时，应在开发计划中包括配置管理计划。在配置管理计划中规定配置标识规则，建立什么样的配置数据库及如何将配置项置于配置管理之下，配置管理人员职责及配置管理活动，采用的配置管理工具、技术和方法。 § 实施变更管理。这是配置管理的一项重要内容，许多软件工程项目因为没有变更管理措施，而导致出现混乱。

§ 实施版本管理和发行管理。

实际上，软件配置时一个动态的概念。制定适当的命名规则是配置标识的重要工作，命名不能任意、随机地进行。对命名的要求有：唯一性：目的在于避免出现重名，以免造成混乱；可追溯性：使命名能够反映命名对象之间的关系。

【10-5】A. ③, B. ②, C. ②, D. ①, E. ④。



“登入”和“检出”处理实现了两个重要的变更控制要素，即存取控制和同步控制。存取控制管理各个软件工程人员存取或修改一个特定的软件配置对象的权限；同步控制用于确保由不同的人所做的并发变更不会产生混乱。

存取和同步控制如图所示。根据经批准的变更请求和变更工程顺序，软件工程人员从项目数据库中检出要变更的配置对象。存取控制功能保证了软件人员具有检出该对象的权限，而同步控制功能则封锁(lock)了项目数据库中的这个对象，使得当前检出的版本在没有被置换前不能再更新它。当然，对这个对象还可以检出另外的副本，但对它也不能更新。软件工程人员在对这种成为基线的对象做了变更，并经过适当的软件质量保证措施和测试之后，把修改版本登入项目数据库，再解除封锁(unlock)。

【10-6】A. ④, B. ②, C. ①, D. ③, E. ③。

软件工程标准的类型是多方面的。它可能包括过程标准，如方法、技术、度量等；产品标准，如需求、设计、部件、描述、计划、报告等；专业标准，如职别、道德准则、认证、特许、课程等；记法标准，如术语、表示法、语言等。根据中国国家标准 GB／T 15538-1995(软件工程标准分类法)规定，软件工程标准可用一张二维的表格来表示。

【10-7】A. ②, B. ①, C. ⑤, D. ③, E. ④, F. ⑤, G. ⑥。

软件工程标准是有层次的，根据软件工程标准制定的机构和标准的适用范围有所不同，它可分为五个级别，即国际标准、国家标准、行业标准、企业规范和项目(课题)规范。

国际标准是由国际联合机构制定和公布，供各国参考的标准，通常冠有 ISO(国际标准化组织)的字样。国家标准是由政府或国家级机构制定或批准，适用于全国范围的标准，如 GB、ANSI、FIPS、BS、DIN、JIS 等。行业标准是由行业机构、学术团体或国际机构制定，并适用于某个业务领域的标准，如 IEEE、GJB、DOD-STD、MIL-S 等。企业规范是由一些大型企业或公司因软件工程工作的需要而制定的适用于本部门的软件工程规范，如 IBM 公司于 1984 年制定的程序设计开发指南，仅供公司内部使用。项目规范由某一科研生产项目，且为该项目专用的软件工程规范，如计算机集成制造系统(CIMS) 的软件工程规范。

【10-8】 ① 错 ② 对 ③ 对 ④ 对 ⑤ 错 ⑥ 对 ⑦ 对 ⑧ 错 ⑨ 错 ⑩ 错。

① 有的标准是强制性的，有的标准是指南性的。虽然国家标准都是由政府或国家级机构制定或批准，适用于全国的标准。这些标准不都是强制性的。例如，GB/T 19000.3－94《质量管理和质量保证标准 第三部分：GB/T 19001－ISO 9001 在软件开发、供应和维护中的使用指南》就是一个建议性的指南，而不是强制性的。

⑤ ISO9000-3 是一个指南性的标准，它针对 ISO 9001 标准的要素按照软件的特点逐条作了解释和说明。它在设计控制的总则中，要求软件开发项目应按照软件生存期模型来组织，要根据所采用的生存期特征来计划和实施过程、活动和任务。但并未涉及具体的开发模式，也未将软件全过程工序从管理角度、合同角度和工程角度划分为三大类。

⑧ 从软件配置管理的角度来讲，在新文档取代旧文档后，管理人员不应删除旧文档。因为文档反映了某些时刻的软件版本信息，旧版本经过修改产生新版本，文档也需随之更新。由于软件的旧版本作为软件配置项仍然保留，反映其状态的旧文档也不能删去。

⑨ 软件配置管理要求把软件生存期中的所有产生的程序、文档、资料等配置项都管理起来，软件开发人员需要修改其中部分文档，必须经过严格的变更控制，不能把主文档的一部分控制在自己手中而不纳入配置管理之下。

⑩ 软件需求分析报告是给开发人员使用的，但其它人员，如管理者和用户等，也需要利用需求分析文档了解软件的需求，参与需求评审和监督软件需求的实现。维护人员在变更软件时也可能要参照需求理解软件、修改软件，甚至修改需求。

【10-9】A. ⑤, B. ③, C. ⑤, D. ②, E. ②。

软件的总体结构应当在概要设计规格说明书中正确定义并给出准确描述。软件的运行环境最初在软件需求规格说明中定义。出错处理设计应在概要设计规格说明中阐明。

初步的用户手册在需求分析阶段开始编写，确认测试计划也应在需求分析阶段开始编写。确认测试有两方面的任务：其一是做有效性测试，确认需求说明书中规定的所有需求是否已正确实现；其二是对所要求的软件配置进行审查，特别是合同中规定应交付的文档进行审查。因为在需求分析阶段已经明确软件的各种功能、性能和其它的质量需求，初步的用户手册也有了，可以针对这些需求和用户手册中的内容，编制如何逐项检查的确认测试计划，当然，这种测试计划只是初步的。测试实施的细节还需在体系结构、用户界面、数据库、出错处理、运行组合等设计完成后才能定下来。

【10-10】① 错 ② 对 ③ 对 ④ 错 ⑤ 错 ⑥ 对 ⑦ 对 ⑧ 错 ⑨ 错 ⑩ 对。

① 可行性研究报告是为管理者提供该项目是否可以立项的决策依据，编写者在提出可能的后选方案并分析各种可行性后应当给出结论，说明该项目是否值得立项，能否获得成功。

④ 编写文档时必须保持各个文档的独立性，不能写“参看××说明书××节”，所以如果各文档有重复的地方时，应从前一阶段的文档中复制过来。

⑤ 用户手册应当使用用户熟知的术语，不应用专业术语。应阐明系统的使用方法，不必详细介绍系统的结构。

⑧ 每个模块的实测结果是单元测试的结果，不应使用需求信息和概要设计(体系结构)信息来做结果比较。

⑨ 软件需求规格说明是对待开发软件系统提出的要求，不包括对软件操作人员和维护人员的教育水平和技术专长的要求。

【10-11】A. ③, B. ①, C. ②, D. ②, E. ①, F. ③, G. ②, H. ④, I. ⑤

保证软件过程质量应包含的工作有：定义过程标准；监控开发过程，报告软件过程。

近年来，软件工程界的专家对过程评估和过程改进表现出浓厚的兴趣。其中有卡内基-梅隆大学 SEI 的 CMM；国际标准化组织(ISO)的 ISO 9001, ISO 9000-3。CMM 的主题是软件过程能力评估，ISO 9001, ISO 9000-3 的主题是建立和维持质量体系。Bell 的 TRILLTUM 的主题是软件过程评估(基于 CMM)。参看表 10.5。

【10-12】A. ④, B. ⑦, C. ⑤, D. ⑥, E. ⑧, F. ②, G. ①, H. ③。

此题是 SEI CMM 中涉及到的一些重要概念。人们用于开发和维护软件及其相关产品的一系列活动称为软件过程；表示(开发组织或项目组)遵循其软件过程所获得的实际结果称为软件过程性能；而描述通过遵循其软件过程能够实现预期结果的程度称为软件过程能力；一个特定软件过程被明确和有效地定义、管理、测量和控制的程度称为软件过程成熟度；表征软件过程成熟度的平台称为软件能力成熟度等级；对软件机构进化阶段的描述，随着软件机构定义、实践、测量、控制和改进其软件过程，软件机构的能力经过这些阶段逐步前进，称为软件能力成熟度模型；互为关联的若干软件实践活动和有关基础设施的一个集合称为关键过程域；对关键过程域的实施起关键作用的方针、规程、措施、活动以及相关基础设施的建立称为关键实践。

【10-13】A. ④, B. ②, C. ②, D. ②, E. ③, F. ①。

基于软件人员能力成熟度模型的评估的依据是成熟度级别上的关键过程域，应当考察特定级别上每个关键过程域的内容是否实施，关键过程域的目标是否达到，同时应考察相应的劳动力条例是否被广泛执行等。一个企业的成熟度级别是所有关键过程域被实施的最低级别。软件企业通过评估结果可以了解关键过程域方面

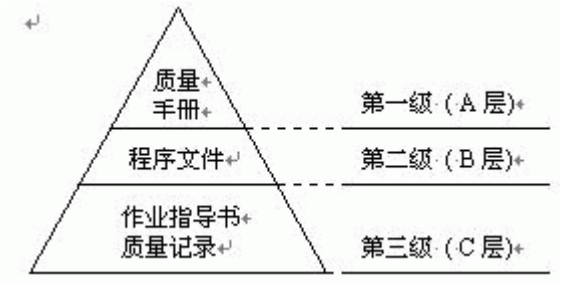
的强项和弱项，明确努力的方向。P-CMM 模型并不禁止处于较低成熟度级别上的企业在需要的时候实施高一级关键过程域的内容。但是，由于每一级都是向上一级进化的基础，所以，跳越成熟度级别的进化是有风险的。基于 P-CMM 的改进工作和其它任何发展项目一样，第一要有计划，第二要跟踪其发展，第三要有专人负责。卡内基－梅隆大学 SEI 的 P-CMM 顾问委员会建议：在软件工程小组内加入人力资源管理人员，以进行基于 P-CMM 的改进工作。这样，带给软件机构管理层的信息就是：一个致力于改善整个软件过程能力的项目强调的是过程、技术和人员，缺一不可。

【10-14】A. ③, B. ④, C. ③, D. ④, E. ②, F. ①。

ISO 9000 族标准是指国际标准化组织中的质量管理和质量保证技术委员会(ISO / TC176)制定的标准，现有 20 个标准，可分为 5 类：质量术语标准，质量保证标准，质量管理标准，质量管理和质量保证标准的选用和实施指南、支持性技术标准。如下表所示。

<table>
<tr><td colspan="4">①. 质量术语标准</td></tr>
<tr><td colspan="4">ISO 8402</td></tr>
<tr><td>④. 标准选用与实施指南</td><td colspan="2">②. 质量保证标准</td><td>③. 质量管理标准</td></tr>
<tr><td>ISO9000 -1. 选择和使用
-2. 实施
-3. 计算机软件
-4. 可信性大纲</td><td colspan="2">ISO9001 设计、开发、生产、安装和服务
ISO9002 生产、安装和服务
ISO9003 最终检验和试验</td><td>ISO9004 -1. 指南
-2. 服务指南
-3. 流程性材料
-4. 质量改进</td></tr>
<tr><td colspan="4">⑤. 支持性技术标准</td></tr>
<tr><td>ISO 10005. 质量计划
ISO 10007. 技术状态</td><td>ISO 10011 -1. 审核
-2. 审核员
-3. 审核管理</td><td>ISO 10012 -1. 测量设备
-2. 测量过程</td><td>ISO 10013. 质量手册</td></tr>
</table>

【10-15】A. ④, B. ⑥, C. ②, D. ①, E. ⑤, F. ②, G. ③, H. ②, I. ①, J. ④。其中，I 和 J 的答案顺序可互换。



质量体系是一种质量管理制度。质量体系是通过若干过程来实现的并有一组质量体系文件，如图所示。过程是将输入转化为输出的一组彼此相关的资源和活动。过程本身具有增值的效果，是一种有效益的行为。质量管理是通过质量体系来实

现的。一个组织机构建立自己的质量体系，并使之有效地运行，使达到质量管理目标的重要手段。

为明确规定、系统地描述质量体系的各项要求，使得相关人员能够理解和方便地实施，有必要建立质量体系文件。质量体系文件大致可分为三个层次，如图所示。

【10-16】A. ③, B. ①, C. ②, D. ④, E. ⑥, F. ③。

ISO 9000-3 是计算机软件机构实施 ISO 9001 的指南性标准。它的指南性主要表现在：

ｉ）对于针对的标准给予特定的说明与解释，即从软件的角度对 ISO 9001 标准的内容给出具体的说明和解释。ISO 9001 提供了 20 个质量体系要素，ISO 9000-3 则对照其每一个要素，逐一作出说明和解释。

ii）指南性的标准不是认证审核的依据。ISO 9001 是认证审核的依据。在认证审核时考察的是申请认证的软件机构贯彻实施 ISO 9001 的情况，而不考察有无违背 ISO 9000-3 条文的情况。事实上，者两个标准文本的叙述语气也是有差别的。在 ISO 9001 的质量体系要素的每一条都强调了所提要求的强制性，使用了“应该(shall)”如何的用语，而在 ISO 9000-3 的各条解释中，由于是建议性指南，普遍使用了“最好(should)”如何的提法。

《ISO／IEC 12207 ：1995 信息技术－软件生存期过程》将整个软件生存期分为 17 个过程，并且对每一个过程按“过程-活动-任务”的三个层次具体做了解释，为我们进一步理解 ISO 9000-3 提供了帮助。

第 5 章 数据库基础知识

数据库（database,DB）技术是计算机软件领域的一个重要分支，产生于 20 世纪 60 年代末。1968 年美国 IBM 公司推出层次模型的 IMS 数据库管理系统，1968 年 10 月美国数据系统语言协会（CODASYL）的数据库存任务组（DBTG）发表了关于网状模型的 DBTG 报告，1970 年美国 IBM 公司的 E。F。CODD 发表论文提出了关系模型，这三件事情奠定了数据库技术的基础。20 世纪 70 年代，网状系统、层次系统得到了广泛的应用，而关系系统处于实验系统和理论研究的阶段，从 20 世纪 80 年代初才开始逐步推出若干关系系统的产品。由于关系模型的理论性强，关系数据库存语言使用的方便性，推动了关系系统的应用和普及，因此从 20 世纪 80 年代末开始，关系系统逐步取代层次系统和网状系统，成为主流产品，本章主要介绍关系模型的基本概念、关系代数操作、关系数据库 SQL 语言、规范理论各数据库的保护等内容。

5.1 关系模型

在数据库技术中，我们用模型的概念描述数据库的结构与语义，对现实世界进行抽象。表示实体类型及实体间联系的模型称为"数据模型"。

关系模型是目前最为流行的一种数据模型，用二维表格结构表示实体集，关键码表示实体间的联系。

5．1．1 关系模型的基本概念

1． 关系模型的基本术语。

在传统的数据为技术、关系模型、典型的关系数据库语言 SQL 中使用的术语有些不同，如表 5.1 所示。

表 5.1 基本术语

数据库技术的术语	关系模型术语	SQL 术语
记录类型	关系模式	基本表
记录	元组	行
文件	关系、实例(表)	基本表、表格
属性、字段、数据项	属性	列

2．键（Key）

键（关键码）是关系模型的一个重要概念，有下列几种键：

（1） 超键：在关系模式中，能惟一标识元组的属性集称为超键（super key）.

（2） 候选键：如果一个属性集能惟一标识元组，且不含有多余属性，那么这个属性集称为候选键（candidate key）。

（3） 主键：关系模式中用户正使用的候选键称为主键（primary key）。一般，如不加说明，键是指主键。

（4） 外键：如果模式 R 中某属性集是其他模式的候选键，那么该属性集对模式 R 而言是外键（foreign key）.

关系着每一个属性都有一个取值范围，这个取值范围称为属性的值域。每一个属性对应一个值域，不同的属性可对应同一个值域。

3.关系的定义

我们可用集合论的观点定义关系（relation）。关系是一个元数为 K（K≥1）的元组的集合。把关系看成一人集合，集合中的元素的元组，每个元组都有 K 个属性值。

如果一个关系的元组个数是无限的，那么称为无限关系；否则称为有限关系。由于计算机存储系统的限制只研究有限关系。

尽管关系与二维表格、传统的数据文件有类似之处，但它们又有区别。严格地说，关系是一种规范化了的二维表格中行的集合。在关系模型中，对关系作如下限制：

（1） 关系中每一个属性值都是不可分解的，也不允许出现重复组；

（2） 关系中不允许出现相同的元组（即不允许出现重复元组）；

（3） 由于关系是一个集合，因此不让用户考虑元组的顺序；

（4） 元组中属性是有序排列的，用户在使用时应考虑到顺序。

4．关系数据库确良的数据体系结构

关系模型遵循数据库的三级体系结构，其名称如下：关系模式、关系子模式和存储模式。

（1）关系模式：数据库的概念模式定义为关系模式的集合。每个关系模式就是记录类型。关系模式的定义包括：模式名、属性名、值域名以及模式的主键。由于不涉及到物理存储方面的细节，因此关系模式仅仅是对数据本身特性的描述。

[例 5．1]教学数据库有三个关系模式，如图 5.1 所示。图 5.2 是这个数据库存的三个具体关系。

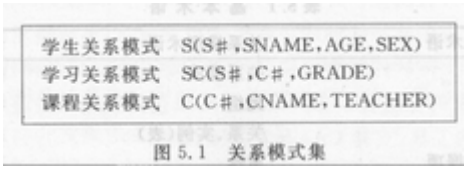


图 5.1 关系模式集

（2） 关系子模式：子模式是用户所用到的那部分数据的描述。除了指出用户用到的数据外，还应指出数据与模式中相应数据的联系，即指出子模式与模式之间的对应性。

S

S#	SNAME	AGE	SEX
S1	Wang	20	M
S4	Wu	19	M
S2	Liu	21	F
S3	Chen	22	M
S8	Dong	18	F

SC

S#	C#	GRADE
S1	C1	80
S3	C1	90
S1	C2	70
S3	C4	70
S3	C2	85
S3	C3	95
S4	C4	70
S8	C3	90

C

C#	CNAME	TEACHER
C2	Maths	Ma
C4	Physics	Shi
C3	Chemistry	Zhou
C1	Database	Li

图 5.2 三个关系

[例 5．2]用户经常用到子模式 G（图 5.3）的数据，这个子模式的构造过程如图 5.4 所示。

成绩子模式 G（S#，SNAME，C#，GRADE）

图 5.3 关系子模式

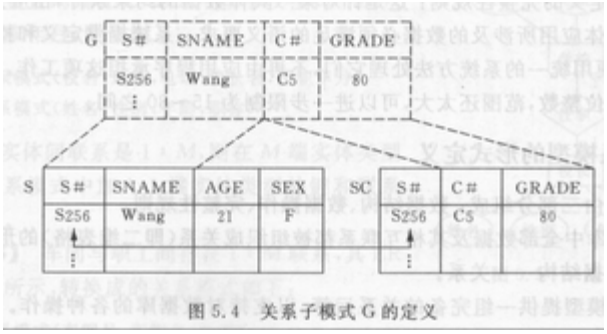


图 5.4 关系子模式 G 的定义

（3）存储模式：关系存储是作为文件看待的，每个元组都是一个记录，由于关系模式有键，因此存储一个关系可用散列方法或索引方法实现。如果关系的元组数目较少（100 个左右），那么也可以用"堆文件"方法实现（即没有特定的次序）。此外，还可对任意的属性集建立辅助索引。

5．1．2 关系模型的三类完整性规则

为了维护数据库中数据与现实世界的一致性，关系数据库的数据与更新操作必须遵循一下列三类完整性规则：

（1） 实体完整性规则（entity integrity rule）：这条规则要求关系中元组在主键的属性上不能有空值。如果出现空值，那么主键值就起不了惟一标识元组的作用。

（2） 引用完整性规则（reference integrity rule）：这条规则要求："不允许引用不存在的元组"。这条规则也称为"参照完整性的规则"。

[例 5.3]在图 5.1 的关系模式集中,S#是模式 S 的主键;在模式 SC 中主键是(S#, C#）而 S#是外键。我们要求关系 SC 中 S#值都应该在关系 S 中出现。如果关系 SC 中有一个元组（S7，C4，80）此时学号 S7 在关系 S 中不存在，我们就认为在关系 SC 中引用了一个不存在的学生实体，这就违反了引用完整性规则。

引用完整性规则的形式定义如下：如果属性集 K 是关系模式 R1 的主键，K 也是另一关系模式 R2 的外键，那么在 R2 的关系中 K 的取值只允许两种可能：空值;等于 R1 的关系中某个元组的 K 值。这条规则使用时，K 大作为主键或外键出现时可以起不同的名字，只要定义在相同的值域上即可。

实体完整性规则和引用完整性规则是关系模型必须满足的规则,应该由系统自动支持。

（3）用户定义的完整性规则：这是针对某一具体数据的约束条件，由应用环境决定。它反映某一具体应用所涉及的数据必须满足的语义要求。系统提供定义和检验这类完整性的机制，以便用统一的系统方法处理它们，不再由应用程序承担这项工作。例如学生年龄的定义为两位整数，范围还太大，可以进一步限制为 15～30 之间.

5.1.3 关系模型的形式定义

关系模型由三部分组成:数据结构、数据操作、完整性规则。

（1） 数据库中全部数据及其相互联系都被组织成关系（即二维表格）的形式。关系模型只是一种数据结构--关系。

（2） 关系模型提供一组完备的关系运算，以支持对数据库的各种操作。关系运算的理论是关系代数和关系演算。关系数据库的数据操纵语言（DML）以关系运算理论为基础实现。

（3） 关系模型有三类完整性规则（在 5.1.2 节已介绍过）。

与其他数据模型相比，关系模型的突出优点如下：

（1） 关系模型提供单一的数据结构形式，具有高度的简明性和精确性。各类用户都能很容易地掌握和运用基于关系模型的数据库系统,使得数据库应用系统的开发效率显著提高。

（2） 关系模型的逻辑结构和数据操作完全独立于数据存储方式 ，具有高度的数据独立性，用户完全不必关心物理存储的细节。

（3） 关系模型建立在坚实的数学基础上。关系运算的完备性、设计规范化理论为数据库技术的成熟奠定了基础。

（4） 关系数据库语言与一阶谓词逻辑的固有内在联系，为以关系数据库为基础的推理系统和知识库系统的研究提供了方便，成为新一代数据库技术不可缺少的基础。

5．1．4 ER 模型向关系模型的转换

ER 模型的主要成分是实体类型和联系类型。

对于实体类型，可以这样转换：将每个实体类型转换成一个关系模式，实体的属性为关系的属性，实体标识符为关系模式的键。

对于联系类型，视不同情况作不同处理。

（1） 若实体间联系是 1：1，可以在两个实体类型转换成的两个关系模式中的任意一个模式内，加入另一个模式的键各联系类型的属性。

[例 5．4]学校与校长间存在 1：1 联系，其 ER 图如图 5。5 所示。转换成关系模型进，学校与校长各成为一个关系模式。如果在查询学校模式时常要查询其校长，那么可在学校模式中加入校长名和任职年月，设计如下：

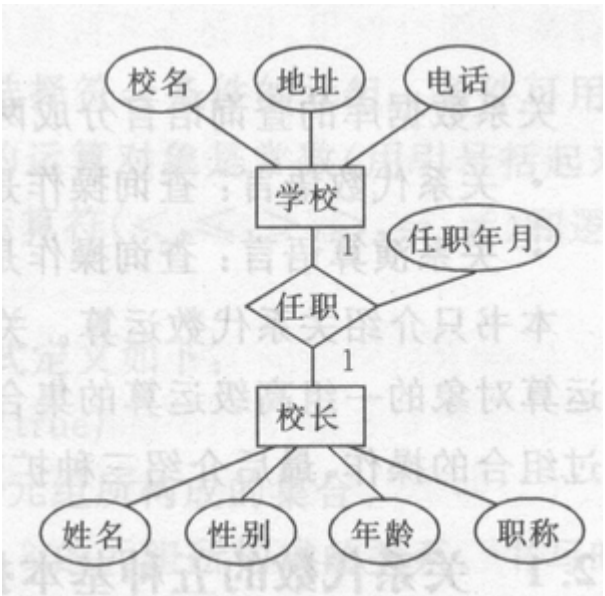


图 5.5 一对一联系

学校关系模式（校名，地址，电话，校长名，任职年月）

校长关系模式（姓名，性别，年龄，职称）

（2） 若实体间联系是 1：M，则在 M 端实体类型转换成的关系模式中加入 1 端实体类型的键和联系类型的属性。

[例 5．5]车间与职工间存在 1：M，其 ER 图如图 5。6 所示，转换成的关系模式如下：
车间关系模式（车间号，车间名，电话）
职工关系模式（工号，姓名，性别，年龄，车间号，聘期）

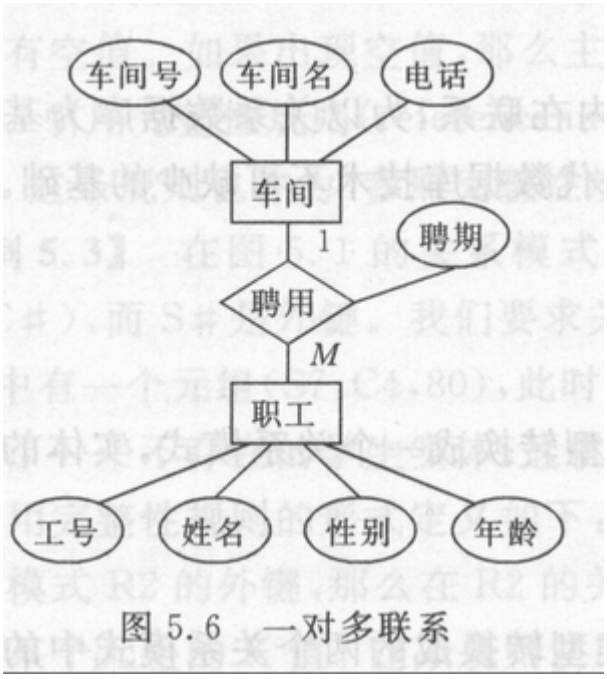


图 5.6 一对多联系

（3） 若实体间联系是 M：N 联系，则将联系类型也转换成关系模式，其属性为两端体类型的键加上联系类型的属性，其键为两端实体类型键的组合。
[例 5．6]学生与课程间有 M：N 联系，其 ER 图如图 5。7 所示，可转换成如下关系模式：
学生关系模式 S（S#，SNAME，AGE，SEX）
成绩关系模式 SC（S#，C#，GRADE）
课程关系模式 C（C#，CNAME，TEACHER）

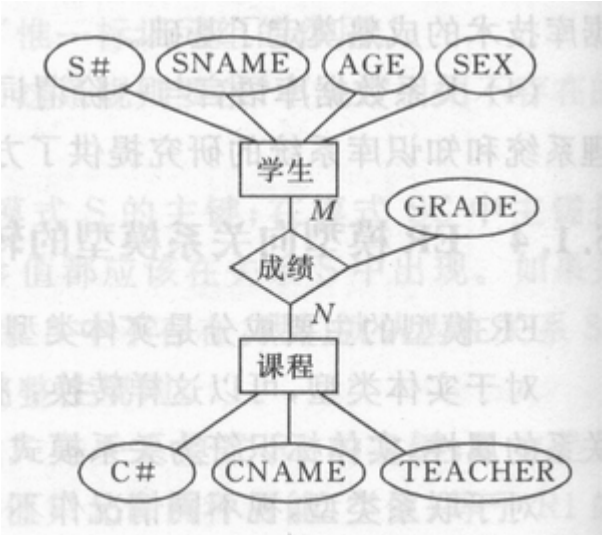


图 5.7 多对多联系

5.2 关系代数

关系数据库的查询语言分成两大类：

- 关系代数语言：查询操作是以集合操作为基础的运算；
- 关系演算语言：查询操作是以谓词演算为基础的运算；

本书只介绍关系代数运算。关系代数是以集合代数为基础发展起来的，它是以关系为运算对象的一组高级运算的集合。本节先介绍关系代数的五种基本操作，再介绍四种经过组合的操作，最后介绍三种扩充的代数操作。

5．2．1 关系代数的五种基本操作。

（1） 并（union）

设关系 R 和 S 具有相同的关系模式，R 和 S 的并是由属于 R 或属于 S 的元组组成的集合，记为 R∪S。形式定义如下：

$R \cup S \equiv \{t \mid t \in R \vee t \in S\}$

式中 t 是元组变量

（2） 差（diference）

关系 R 和 S 具有相同的关系模式，R 和 S 的差是由属于 R 但不属于 S 的元组组成的集合，记为 R-S。形式定义如下：

$R-S \equiv \{t \mid t \in R \wedge t \notin S\}$

（3） 笛卡尔积（cartesian product）

设关系 R 和 S 元数分别为 r 和 s。定义 R 和 S 的笛卡尔积是一个（r+s）元的元组集合，每人个元组的前 r 个分量来自 R 的一个元组，后 s 个分量来自 S 的一个元组，记为 R×S。形成定义如下：

$R \times S \equiv \{t \mid t=\langle t^r, t^s \rangle \wedge t^r \in R \wedge t^s \in S\}$

式中 t^r，t^s 中的 r,s 为上标。若 R 为 m 个元组，S 有 n 个元组，则 R×S 有（mn）个元组。

（4）投影（projection）

该操作对关系进行垂直分割，消去某些列，并重新安排列的顺序，再删去重复元组。
设关系 R 是 K 元关系，R 在其分量 Ai1,……,Aim（m≤k,i1,……,im 为 1 到 K 互不相同的整数）上的投影为 л i1,……im（R）表示，综是一个 m 元元组的集合，形式定义如下：

$л\, i1, \cdots\cdots, im(R)=\{t \mid t=\langle ti, \cdots\cdots, tim \rangle \wedge \langle t1, \cdots\cdots, tk \rangle \in R\}$

例如 л 3，1（R）表示在关系 R 中取第 3，1 列，组成新的关系，新关系的列按顺序是原关系的第 3，1 列。如果 R 的每列标上属性名，那么操作符 л 的下标处也可以不写列的序号，写上属性名。例如关系 R（ABC），$л_{c,A}(R)$ 和 $л_{3,1}$（R）是等价的。

（5）选择（selection）

这个操作是根据某些条件对关系作水平分割，即选择符合条件的元组。条件可用命题公式 F（即计算机语言中的条件表达式）表示，F 中的运算对象是常数（用引号括起来）或元组分量（属性名或列的序号）运算符有算术比较运算符（<,≤,>,≥,=,≠）和逻辑运算符（∧，∨，┐）。
关于 R 关于公式 F 的选择运算用 бF（R）表示，从形式定义上如下：

$Б F(R) \equiv \{t \mid t \in R \wedge F(t)=ture\}$

Б 为选择运算符，бF（R）表示从 R 中挑选满足公式 F 的元组构成的集合。

例如 б2>'3'（R）表示从 R 中挑选第二个分量值大于 3 的元组所构成的关系。书写时，常量用引号括起来，而属性序号或属性名不要用引号括起来。

整理人：刘海云
整理时间：2003.05

[例 5．7]图 5.8 有两个关系 R 和 S。图 5.9 的（a），（b）分别表示 R∪S 和 R-S。（c）表示 R×S，此处 R 和 S 的属性名相同，就在属性名前标上相应的关系名，例如 R.A,S.A 等。（d）表示 лA,C(R)即 л1,3（R）。（e）表示 БB='b'(R).



图 5.9 关系代数操作

上面五种操作是关系代数最基本的操作，也是一个完备的操作集。

5．2．2 关系代数的四种组合操作

在关系代数中还可以引进其他许多操作，这些操作不增加语言的表达功能，可从上面五种操作组合推出，但是在实际使用中极为有用。

（1）交（intersection）

关系 R 和 S 具有相同的关系模式，R 和 S 的交是由属于 R 又属于 S 的元组构成的集合，记为 R∩S 。形式定义如下：

R∩S≡{t|t∈R∧t∈S}

显然 R∩S=R-（R-S）， 或者 R∩S=S-（S-R）。例如图 5。8R∩S 的结果是只有一个元组（daf）。

联接(join)联接有两种：θ联接和 F 联接。

θ联接操作是从关系 R 和 S 的笛卡积中选取属性值之间满足某一θ操作的元组，记为 $R\triangleright\!\lhd_{i\theta j}S$。

这里 i 和 j 分别是关系 R 和 S 中第 i 个、第 j 个属性，θ是算术比较运算符。形式定义如下：

$$R\triangleright\!\lhd_{i\theta j}S\equiv \mathrm{B}_{i\theta(r+j)}(R\times S)$$

这里 r 是关系 R 的元数，$R\triangleright\!\lhd_{i\theta j}S$ 是在 R 和 S 的笛卡尔积中挑选第 I 个分量和第（r+j）个分量满足θ运算的元组。如果θ是等号“=”，那么这个联接操作称为“等值联接”。

F 联接操作是从关系 R 和 S 的笛卡尔积中选取属性值满足某一公式 F 元组，记为 $R\triangleright\!\lhd_{F}S$

这里 F 是形为 F1∧F2∧……∧Fn 的公式，每个 F_p 是开为 $i\theta j$ 的式子，而 i 和 j 应分别为 R 和 S 中第 i 个分量和第 j 个分量。

[例 5. 8]图 5.10 的（a），（b）是关系 R 和 S，（c）是 $R\triangleright\!\lhd_{2<1}S$

其中 $\triangleright\!\lhd_{2<1}$ 也可以写成 $\triangleright\!\lhd_{B<D}$，但要注意 $R\triangleright\!\lhd_{2<1}S=\mathrm{B}_{z<4}(R\times S)$。（d）是 $R\triangleright\!\lhd_{2<1\wedge 1\geqslant 2}S$ 的值。

（3）自然联接（natural join）

两个关系 R 和 S 的自然联接用 $R\triangleright\!\lhd S$ 表示，具体计算过程如下：

①计算 R×S；

②设 R 和 S 的公共属性是 A1，……AK，挑选 R×S 中满足 $R.A1=S.A1,\cdots\cdots,R.A_r=S.A_r$ 的那些元组；

③去掉 $S.A1,\cdots\cdots,S.A_r$ 列。

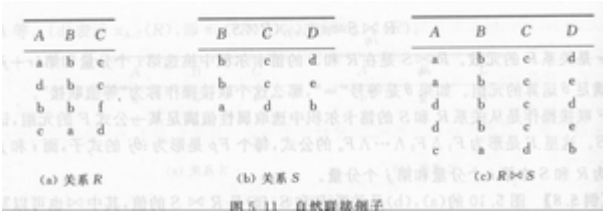
因而且　　可以用下列形式定义：

$$R\triangleright\!\lhd S\equiv \Pi_{i1,\ldots,im}(\mathrm{B}_{R.A1=S.A1\wedge\ldots\wedge R.A_r}(R\times S))$$

[例 5. 9]图 5.11 表示关系 R 和 S 的自然联接，这里

$$R\triangleright\!\lhd S=\Pi_{A,R.B,R.C,D}(\mathrm{B}_{R.B=S.B\wedge R.C=(R\times S)})$$

自然联接是构造新关系的有效方法，是关系代数中常用的一种运算。利用投影、选择和自然联接操作可以任意地分割和构造关系。一般自然联接使用在 R 和 S 有公共属性的情况中。如果两个关系没有公共属性，那么它们的自然联接就变化为笛卡尔积。



（4） 除法（division）

设两个关系 R 和 S 的元数分别为 r 和 s（r>s>0），那么 R÷S 是一个（r-s）元的元组的集合。（R÷S）是满足下列条件的最大关系：其中每个元组 t 与 S 中每个元组 u 组成的新元组＜t,>u 必在关系 R 中.为方便,这里我们假设 S 的属性为 R 中后 s 个属性.形式如下:

[例 5.10]图 5.12 是两个关系做除法的例子.



A	B	C	D
a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
e	d	e	f
a	b	d	e

(a) 关系 R

C	D
c	d
e	f

(b) 关系 S

A	B
a	b
e	d

(c) R÷S

图 5.12 除法操作的例子

在关系代数中,由五种基本代数操作经过有限次复合的式子称为关系代数运算表达式.表达式的运算结果仍是一个关系.我们可以用关系代数表达式表示各种数据查询和更新处理操作.

5.2.3 关系代数运算的应用实例

[例 5.11]学生与课程数据库中有下列三个关系模式:

学生关系模式 S(S#,SNAME,AGE,SEX)

成绩关系模式 SC(S#,C#,GRADE)

课程关系模式 C(C#,CNAME,TEACHER)

S 的属性分别表示学生的学号、姓名、年龄和性别，SC 的属性分别表示学号、课程号成绩，C 的属性分别表示课程号、课程名、任课教师姓名。

试用关系代数表达式表示每个查询语句：

① 检索学习课程号为 C2 的学生学号与成绩

Π s#,GRADE(Б C#='C2'(SC))

也可以写成

Π 1,3(Б 2='C2'(SC))

② 检索学习课程号为 C2 的学生学号与姓名

Πs#,SNAME(Б C#='C2'(S⋈ SC))

③ 检索选修课程名为 Maths 的学生学号与姓名

Πs#,SNAME(Б CNAME='Maths'(S⋈ SC⋈ C))

④检索选修课程号为 C2 和 C4 的学生学号

ΠS#(Б C#='C2'∨ C# ='C4'(SC))

⑤检索至少选取修课程为 C2 和 C4 的学生学号

Π1(Б 1=4∧2='C2' ∧5='C4'(SC×SC))

这里（SC×SC）表示关系 SC 自身相乘的笛卡尔积操作

⑥检索不学 C2 课程的学生姓名与年龄

ΠSNAME,AGE(S)- ΠSNAME,AGE （Б C#='C2'(S⋈ SC))

⑦检索学习全部课程的学生姓名

ΠSNAME((S⋈ (ΠS#,C#(SC)÷ΠC#(C)))

⑧检索所学课程包含学生 S3 所学课程的学生学号

ΠS#,C#(SC)÷ΠC#(Б S#='S3'(SC)

5.2.4 扩充的关系代数操作

为了在关系代数操作时，多保存一些信息，就引进了"外联接"和"外部并"两种操作。

（1） 外联接（outerjoin）

两个关系 R 和 S 在自然联接时，选择两个关系在公共属性上值相等的元组构成新关系的元组。此时，关系户 R 中某些元组有可能在 S 中不存在公共属性上值相等的元组，造成 R 中这些元组的值在操作时舍弃了。同样 S 中某些元组也有可能被舍弃。

如果 R 和 S 做自然联接时，把该舍弃的元组也放在新关系中，在新增加的属性上填上空值（null），那么这种操作称为"外联接"操作，用符号 R]×[S 表示。

如果 R 和 S 做自然联接时，只反 R 中原该舍弃的元组放到新关系中，那么这种操作称为"左外联接"操作，用符号 R]×|S 表示。

如果 R 和 S 做自然联接时，只把 S 中原该舍弃的元组放到新关系中，那么这种操作称为"右外联接"操作，用符号 R|×[S 表示。

[5.12]图 5.13 中（a），（b）是关系 R 和 S，（c）为 R 和 S 的自然联接操作 R⊳⊲S，（d）为外联接操作 R]×[S，（e）为左外联接操作 R]×|S，（f）为右外联接操作 R]×|S。



图 5.13 外联接操作的例子

（2） 外部并（outer union）

前面定义两个关系的并操作时，要求 R 和 S 具有相同的关系模式。如果 R 和 S 的关系模式不同，构成的新关系的属性由 R 和 S 的属性组成（公共属性只取一次），新关系的元组由属于 R 或属于 S 的元组构成，此时元组应在增加的属性上填上空值，那么这种操作称为"外部并"操作。

[例 5. 13]图 5.14 就是图 5.13 中关系 R 和 S 做外部并操作后的结果。

A	B	C	D
a	b	c	null
b	b	f	null
c	a	d	null
null	b	c	d
null	b	c	e
null	a	d	b
null	e	f	g

图 5.14 外部并操作的例子

在分布式数据库中经常要用到下面一种"半联接"操作。

（3） 半联接（semijoin）

关系 R 和 S 的半联接操作记为 R∝S，定义为 R 和 S 的自然联接在关系 R 的属性集上的投影，即

$R \propto S \equiv \Pi R(R \triangleright\!\triangleleft S)$

此外的脚步标 ΠR 表示关系 R 的属性集。

也可以用另一种方法计算 R∝S：先求出 S 在 R 和 S 的公共属性集上的投影，R 和这个投影的自然联接，即 $R \propto S = R \triangleright\!\triangleleft \Pi R \cap S(S)$。显然半联接的交换律是不成立的，即 R∝S≠S∝R。
[

例 5. 14]图 5.15 是两个关系做自然联接和半联接的例子。



图 5.15 自然联接和半联接的例子

5.3 关系数据库 SQL 语言

结构化查询语言（structured query language,SQL）是 20 世纪 70 年代初提出的，现已成为关系数据库的标准语言。SQLJ 是集 DDL、DML 和数据控制功能于一体的数据为语言。SQL 的 DML 是介乎于关系代数和关系演算之间的一种语言。由于 SQL 使用方便、功能丰富、简洁易学，很快得到应用和推广。例如关系 DBMS 的产品 ORACLE，SQL/DS，DB2 SYBASE 等都以 SQL 作为数据库语言。同时，其他数据库产品的厂家也纷纷推出各自的 SQL 的软件或者与 SQL 接口的软件。很快，SQL 语言得到了整个计算机界的认可。

国际标准化组织（ISO）在 1992 年推出了较完善的标准--SQL2（也称为 SQL92）。SQL 的标准化工作还在继续，新的标准已命名为 SQL3，在 1993 年推出 SQL3 的核心部分标准，包括面向对象和许多新的数据库概念。现在还在不断征求意见和进行修改。

5．3．1SQL 数据库的数据体系结构

SQL 数据库的数据体系结构基本上也是三级结构（图 5.16），但术语与传统关系模型术语不同。在 SQL 中，关系模式称为"基本表"（base table），存储模式称为"存储文件"（stored file），子模式称为"视图"（view），元组称为"行"(row)，属性称为"列"（column）。

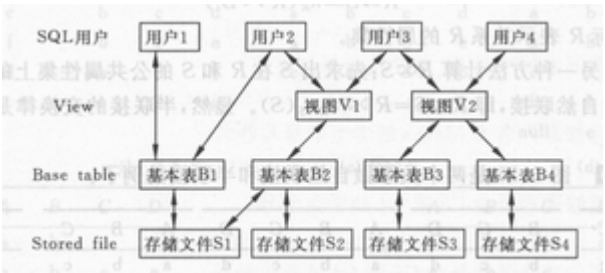


图 5.16 SQL 数据库的体系结构

5．3．2 SQL 语言的组成

一个 SQL 数据库是表（table）的汇集，它用一个或多个 SQL 模式定义。

一个 SQL 表由行集构成每行是列的序列，每列对应一个数据项。

一个表或者是一个基本表，或者是一个视图。视图是由若干基本表或其他视图构成的表的定义。

一个基本表可以跨一个或多个存储文件，一个存储文件也可存放一个或多个基本表。每个存储文件与外部存储器上的物理文件对应。

用户可以用ＳＱＬ语句对视图和基本表进行查询等操作。在用户看来，视图和基本表是一样的，都是关系（即表格）。

SQL 用户可以是应用程序，也可以是终端用户。SQL 语句可嵌入在宿主语言诉程序中使用，宿主语言用 FORTRAN，COBOL，PASCAL，PL/I，C 和 Ada 语言。SQL 用户也能作为独立的用户接口，供交互环境下的终端用户使用。

SQL 包括了所有对数据库操作，主要有四个部分：

（1） 数据定义：这一部分又称为"SQL DDL"用于定义 SQL 模式、基本表、视图、索引等。

（2） 数据操作：这一部分又称为"SQL DML"，分为数据查询和数据更新两大类操作，基中数据更新包括插入、删除和更新三种操作。

（3） 数据控制：基本表和视图的授权、完整性规则的描述和事务控制语句等。

（4） 嵌入式 SQL 语言的使用规定。

上述大部分内容在本节介绍，有些完整性规则、事务控制等语句将在 5.5 节"数据库保护"中描述和介绍。

5．3．3 SQL DDL

SQL 的 DDL 包括对 SQL 模式、基本表、视图的定义和撤销操作。有关视图的内容将在 5.3.5 节介绍。

1． 模式的定义和撤销

（1） 模式的定义

在 SQL2 中，一个数据库模式（即 SQL 模式）被定义为基本表的集合。一个 SQL 模式由模式名和模式拥有者的用户名或帐号确定，并包含模式中每一个元素（基本表、视图、索引）的定义。定义 SQL 模式实际上是定义了一个存储空间。模式定义可用"CREATE SCHEMA"语句实现，例如下面的语句定义了一个学生课程的 SQL 模式：

CREATE SCHEMA ST_CO AUTHORIZETION LISMITH

该模式的拥有者为 LISMITH

SQL2 中还有一个"目录"概念。目录（CATALOG）是 SQL 环境中所有模式的集合。目录是由一个特殊的模式"INFORMATION_SCHEMA"组成，负责提供所有模式描述的元素的信息。这些信息除了基本表、视图、索引等的定义外，还包括完整性规则等内容。同一个目录的模式之间还可以共享域定义一类信息。

（2） 模式的撤销

当一个SQL模式及其所属基本表、视图等元素都不需要时，可以用"DROP SCHE-MA"语句撤销这个SQL模式。撤销的方式有两种CASCADE（连锁式）和RESTRICT（约束式）。

例如，当SQL模式ST_CO及其下属的所有基本表、域、视图等都不需要时，可用下列CASCADE方式实现：

DROP SCHEMA ST_CO CASCADE

在上述语句中，CASCADE可以用RESTRICT替代，此时表示SQL模式中没有任何元素时，才能被撤销，否则拒绝执行。

2. SQL2提供的基本数据类型

CHAR(n) 长度为n的定长字符串

VARCHAR(n) 具有最大长度为n的变长字符串

INT 长整数（也可以写成INTEGER）

SMALLINT 短整数

NUMERIC(p,d) 定点数，由p位数字组成（不包括符号、小数点），小数点后面有d位数字

REAL 浮点数（精度取决于机器）

DOUBLE PRECISION 双精度浮点（精度取决于机器）

FLOAT（n） 浮点数（精度至少为n位数字）

DATE 日期，形成 YYYY-MM-DD

TIME 时间，形为HH：MM：SS

SQL允许在上面列出的域上执行比较操作，便算术操作只限于数值类型。SQL还提供一种时间间隔（INTERVAL）的数据类型。例如两个日期类型的值的差，就是一个间隔类型的值。一个日期类型的值加上（或减去）一个间隔类型的值，就可得到另外一个日期。

SQL2允许用户使用"CREATE DOMAIN"语句定义新的域，例如下面语句定义了一个新的域PERSON-NAME：

CREATE DOMAIN PERSON_NAME CHAR（10）

以后就可以把 PERSON_NAME 作为基本数据类型一样看待，用户在定义属性时可以采用 PERSON_NAME 作为数据类型。

3．基本表的定义、修改和撤销

（1） 基本表的定义

可以用下列语句实现：

CREATE TABLE 模式名,关系名

（属性名 类型,

……

完整性约束，

……）

定义基本表，需指出放在哪个模式中，下面为简单起见，模式名都省略不写。每人属性的类型可以用基本数据类型，也可以是用户预先定义的域名。完整性约束主要有三种子名：主键句(PRIMARY KEY)、检查子句(CHECK)和外键子句(FOREIGN KEY）。

下面举例说明。

[例 5．15]图 5.1 的教学数据库中三个关系模式可用下列语句实现：

CRATE TABLE S

（S# CHAR（4），

SNAME CHAR（8） NOT NULL，

AGE SMALLINT,

SEX CHAR（1），

PRIMARY KEY（S#））

在 SQL 中允许属性值是空值，如果注上了"NOT NULL"，则表示该属性值不允许空。定义了 S#值不允许空。

CREATE TABLE C

（C# CHAR（4），

```
CNAME CHAR（10） NOT NULL，

TEACHER CHAR（8），

PRIMARY KEY（C#））

CREATE TABLE SC

（S# CHAR（4），

C# CHAR（4），

GRADE SMALLINT，

PRIMARY KEY（S#，C#），

FOREIGN KEY（S#）REFERENCES S（S#），

FOREIGN KEY（C#）REFERENCES C（C#），

CHECK（GRADE IS NULL）OR

（GRADE BETWEEN 0 AND 100））
```

在基本表SC中,主键是(S#,C#),还定义了两个外键,并指出外键S#和基本表S中S#相对应,外键C#和基本表C中C#相对应.此外对应的属性恰好同名,实际上也可以不同名,只要拽出其对应性即可.外键体现了关系数据库的引用完整性.SC的定义中还使用检查子名,指出成绩GRADE值或者为空,或者就在0～100之间.

CHECK子句中的条件可以很复杂,甚至可嵌有SELECT查询语句.例如,我们要定义一个男学生的学习成绩基本表SC_M(S#,C#,GRADE),那么,可在SC_M的定义末尾再加上一个CHECK子句:

```
CHECK(S# IN(SELECT S# FROM S WHERE SEX='M'))
```

这样在往SC-M中插入学生的学习成绩时,都要到基本表S中检查此学生是否为"男性",这就保证了数据库的完整性.但是应该注意,带复杂条件的CHECK的测试是很花时间的,会降低系统的运行效率。

（2） 基本表结构的修改

在基本表建立产使用一段时间后，可以根据实际需要对基本表的结构进行修改，即增加新的属性或删除属性。

① 增加属性可以用"ALTER…ADD…"语句

【例 5.16】要在学生基本表 S 中增加一人地址 ADDRESS 属性,可用下列语句:

ALTER TABLE S ADD ADDRESS VARCHAR(30)

应注意,新增加的属性不能定义为"NOT NULL"基本表在增加了一个属性后，原来的元组在新增加的属性上的值均被置成空值。

② 删除属性可以用"ALTER …DROP…"语句

[例 5. 17]要在基本表 S 中删除属性"年龄"，可用下列语句实现，但必须指出删除的方式是

CASCADE 还是 RESTRICT：

ALTER TABLE S DROP AGE CASCADE

或

ALTER TABLE S DROP AGE RESTPRICT

这里 CASCADE 方式表示在基本表 S 中删除属性 AGE 时,所有引用到属性 AGE 的视图或约束也要一起自动地被删除，否则拒绝删除操作。

（3）基本表的撤销

在数据库中撤消已存在的基本表,可以用 DROP 语句,但必须指出方式是 CAS-CADE 还是 RESTRICT。

[例 5.18]撤销基本表 S，可用下列语句：

DROP TABLE S CASCADE

或

DROP TABLE S RESTRICT

这里 CASCADE 和 RESTRIC 的语义与例 5.17 中的解释相同。

5．3．4SQL DML 的数据查询语句

1． SELECT 句型

在关系代数中最常用的式子是"投影选择联接表达式"：

ΠA1,A2,..,An(БF(R1×R2×…×Rm))

此处 R1，R2，……，Rm 为基本表，F 是公式，A1，A2，…，An 为属性。

针对这个表达式，SQL 设计了 SELECT 句型：

SELECT A1，A2，……，An

FROM R1，R2，……，Rm

WHERE F

这个句型从关系代数表达式演变而来，但 WJERE 子句中条件表达式 F 要比关系代数表厉害式中的公式更灵活。

在 WHERE 子句的表达式 F 中可出现下列操作符和运算特点：

（1） 算术比较运算符：<,<=,>,>,>=,=,<>.

（2） 逻辑运算符:AND,OR,NOT.

（3） 集合运算符:UNION(并),INTERSECT(交),EXCEPT(差).

（4） 集合成员资格运算符:IN,NOT,IN.

（5） 谓词:EXISTS(存在量词),ALL,SOME,UNIQUE②.

（6） 聚合函数:AVG(平均值),MIN(最小值),MAX(最大值),SUM(和),COUNT(计数).

（7） F 中运算对象还可以是另一个 SELECT 语句,即 SELECT 语句可以嵌套.

上面列出了 WHERE 子句中可出现的主要操作.由于 WHERE 子句中的条件表达式可以很复杂,因此,SELECT 句型能表达的语义远比那个"投影选择联接表达式"复杂得多,SELECT 语句能表达所遥关系代数表达式.

[例 5.19]在教学数据库的基本表 S,SC,C 中检索数据.

① 检索学习课程号为 C2 的学生学号与姓名

这个查询要从两个基本表中检索数据,因而有多种写法.

第一种写法(联接查询):

SELECT S.S#,SNAME

FROM S,SC

WHERE S.S#=SC.S#

AND C#='C2'

第二种写法(嵌套查询):

SELECT S#,SNAME SELECT S#,SNAME

FROM S 或 FROM S

WHERE S# IN WHERE 'C2' IN

(SELECT S# (SELEXT C#

FROM SC FROM SC

WHERE C# ='C2' WHERE SC.S#=S.S#

此处右边写法中的内查询称为"相关子查询",子查询中的查询条件依赖于外层查询中的值，而且子查询的处理不只一次，要反复求值，以供外层查询使用。

在联接查询中，要对 S 和 SC 做笛卡尔积，因此嵌套查询的效率要比联接查询的效率高，早期的 SQL 语言要求用户尽可能写嵌套查询，而现在关系 DBMS 则允许用户书写简单的联接查询形式，这是因为 DML 的编译程序具有优化查询的功能。

嵌套查询中的谓词 IN 也可以写成"=SOME"形式。

第三种写法（使用存在量词的嵌套查询）：

SELECT S#, SNAME

FROM S

WHERE EXISTS

（SELECT *

FROM SC

WHERE SC.S# =S.S#

AND C#='C2'

② 检索不学 C2 课程的学生姓名与年龄

此例只能用鞭套查询或存在量词方式书定写。

```
SELECT SNAME, AGE
FROM S
WHERE S# NOT IN
 (SELECT S#
FROM SC
WHERE C#='C2'
或 SELECT SNAME, AGE
FROM S
WHERE NOT EXISTS
 (SELECT *
FROM SC
WHERE SC.S#= S.S#
AND C#='C2')
```

或

```
SELECT SNAME, AGE
FROM S
WHERE 'C2' NOT IN
 (SELECT C#
  FROM SC
  WHERE SC.S#=S.S#)
```

③ 检索学习全部课程的学生姓名

```
SELECT SNAME

FROM S

WHERE NOT EXISTS

（SELECT *

FROM C

WHERE NOT EXISTS

（SELECT *

FROM SC

WHERE SC.S#=S.S#

AND SC.C#=C.C#））
```

2. SELECT 句型完整的句法

SELECT 语句由六个子句组成，便只有前两个子句是必不可少的，完整的句法如下：

```
SELECT 目标表的列名或列表达式序列

FROM 基本表或（和）视图序列

[WHERE 条件表达式]
```

[GROUP BY 列名序列

[HAVING 组条件表达式]]

[ORDER BY 列名[序]……]

此处[]表示该子句可用，也可不用，视实际情况而定。整个 SELECT 语句语义如下：从 FROM 子句中列出的表中，选取满足 WHERE 子句中给出的条件表达式的元组，然后按 GROUP 子句中给定列的值分组，再提取满足 HAVING 子句中组条件表达式的那些组，按 SELECT 子句给出的列名或列表表达式求值输出。ORDER 子句是对输出的目标表进行排序，可附加 ASC（升序）或 DESC（降序）。

[例 5．20]对基本表 S，SC，C 进行查询。

① 统计每一年龄行修课程的学生人数。

SELECT AGE，COUNT（DISTINCT S.S#）

FROM S，SC

WHERE S.S#=SC.S#

GROUP BY AGE

此处 SELECT 子句中一定要加 DISTINCT，否则求得的是"每一年龄选修课程的学生人次数"。

② 求表 S 中男同学的每一年龄组（超过 50 人）有多少人？要求查询结果按人数升序排列，人数相同按年龄降序排列。

SELECT AGE ，COUNT（S#）

FROM S

WHERE SEX='M'

GROUP BY AGE

HAVING COUNT（*）>50

ORDER BY 2，AGE DESC

3． SELECT 句型中的各种规定

这里介绍 SELECT 语句书定时各种具体的约定和规定.

(1) SELECT 子句中的规定.

在 SELECT 语句查询结果中是允许出现重复元组的.如果我们不允许出现重复元组,那么要在关键字 SELECT 后面加上关键字 DISTINCT.如果允许出现重复元组,那么可在 SELECT 后加关键字 ALL,但 ALL 是可以省略不写的.

(2) 条件表达式中的算术比较操作

条件表达式中可以出现逻辑运算符(AND,OR,NOT)和算术比较运算符(<,<,=,>,>,=,=,<,>),也可用"BETWEEN18 AND 20"代替。

（3） 属性和基本表的改名操作

在查询输出的属性名和基本表的属性名不一致时，可以根据用户的需要，用"旧名 AS 新名"形式改名。

有时一个基本表在 SELECT 语句中要使用多次，即这个表被多次调用，为区别不同场合，应该每次使用时标不同的名字，即把表定义为一个元组变量，这种方式称为"基本表的改名"操作，也是"旧名 AS 新名"形式。

[例 5. 21]在表 S 中检索男学生的出生年份，输出的列名 STUDENT-NAME 和 BIRTH-YEAR，可用下列语句实现：

```
SELECT SNAME AS STUDENT-NAME，1998-AGE BIRTH-YEAR

FROM S

WHERE SEX='M'
```

在实际使用时，AS 字样可缺省。

（4） 字符串的匹配操作

条件表达式中字符串匹配操作可用操作符 LIKE。在表达式中的通配符如下：

%（百分号）：与零个或多个字符组成的字符串匹配。

_（下划线）：与单个字符匹配。

例如"姓名以字符 D 打头的学生姓名"这个条件可写成"SNAME LIKE'D%'"。

（5） 集合的并、交、差操作

在两个子查询结果的结构完全一样时，我们可以让这两个子查询执行并、交、差操作。

（SELECT 查询语句 1） （SELECT 查询语句 1）

UNION UNION ALL

（SELECT 查询语句 2） （SELECT 查询语句 2）

这两个操作都是执行并操作，但左边操作返回的结果中消除了重复元组，而右边带 ALL 的操作返回的结果中消除重复元组。

交操作 INTERSECT、差操作 EXCEPT 与上述情况类似。

（6） 空值的处理 SQL 中允许属性值为空，用关键字 NULL 表示空值。

例如，测试"年龄为空值"和条件可用"AGE IS NULL"表示，测试"年龄为非空值"的条件可用"AGE IS NOT NULL"表示。

空值的存在增加了算术操作和比较操作的复杂性。SQL 规定，涉及到+，-，*，/的算术表达式中有一个值是空值时，表达式的值也是空值。涉及到空值的比较操作的结果是"false"

SQL2 规定，涉及到倾家荡产值的比较操作的结果的值是"unknown"，这是三值逻辑中的一个值。SQL2 允许用户测试比较的值是否为"unknown"。但在其他情况中，SQL2 把 unknown 值处理为 false 值。

在聚合函数中遇到空值时，除了 COUNT（*）外，都跳过空值而去处理非空值。

（7） 集合的比较操作

在 WHERE 子句中，允许进行集合的比较操作，有下列四种：集合成员资格比较、集合成员的算术比较、空关系的测试、重复元组的测试。

① 集合成员资格的比较

形式如下：

元组 IN（集合）

元组 NOT IN（集合）

这里的集合可以是一个 SELECT 语句，或元组的集合，但其元组的结构应与前面的元组的结构相同。IN 操作表示：如果元组在集合内，那么其逻辑值为 true，否则为 false；NOT IN 操作表示：如果元组不在集合内，那么其逻辑值为 ture，否则为 false。

② 集合成员的算术比较

形式如下：

元组θ SOME（集合）

元组θ ALL（集合）

这里θ是算术比较运算符：<,<=,>,>=,=,<>. "θ SOME"操作表示左边的那个元组与右边集合中至少一个元素满足θ运算。"θ ALL"操作表示左边的元组与右边的集合中每一个元素满足θ运算。

这里涉及到两个元组的比较，例如（a1,a2）<=（b1,b2）的意义与（a1<b1）OR（（a1=b1）AND(a2<=b2)等价。两个元组相等，则要求其对应的属性值都相等。

应注意："=SOME"与"IN"等价，"<>ALL"与"NOT IN"等价。

[例 5. 22]在基本表 SC 中检索平均成绩最高的学生学号。

```
SELECT S#

FROM SC

GROUP BY S#

HAVING AVG（GRADE）>=ALL（SELECT AVG（GRADE）

FROM SC

GROUP BY S#
```

在 SQL 中，不允许聚合函数进行复合运算，因此不能写成"SELECT MAX（AVG（GRADE））"这种形式。

③ 空关系的测试

可以用谓词 EXISTS 测试一个集合是否为非空，或空。其形式如下：

EXISTS（集合）

NOT EXISTS（集合）

前一个操作，当集合非空时（即至少存在一个元素），其逻辑值为 true，否则为 false。后一个操作，当集合中存在重复元组进，其逻辑值为 true,否则为 false.

（8） 导出关系的使用

SQL2 允许在 FROM 子句中使用子查询。如果在 FROM 子句中使用了子查询，那么要给这个子查询起个表名和相应的属性名。

[例 5．23]在基本表 SC 中检索平均成绩超过 80 分的学生学号和平均成绩。

```
SELECT S#，AVG（GRADE）

FROM SC

GROUP BY S#

HAVING AVG（GRADE）>80
```

此查询也可以把检索每个学生的平均成绩的子查询写在 FROM 子句中：

```
SELECT S#，AVG_GRADE

FROM（SELECT S#，AVG（GRADE）

FROM SC

GROUP BY S#

AS RESULT（S#，AVG_GRADE）

WHERE AVG_GRADE>80
```

在此处 RESULT 是 FROM 子句中子查询结果的名字。

4． 基本表的联接操作

在 80 年代，人们对 SQL 提出了批评，认为 SQL 无直接的联接或自然联接的操作。SQL2 吸收了这个意见，用较为直接的形式表示各式各样的联接操作，这些操作都有在 FROM 子句中以直接的形式给出。限于篇幅，举例稍作说明。

[例 5．24]没有关系 R 和 S，如图 5.17 的（a）和（b）。下面有三个 R 和 S 做联接的例子：

① R NATURAL LEFT OUTER JOIN S

此式表示关系 R 和 S 执行自然联接，并且是左外联接。结果如图 5.17 的（c）所示。

② R LEFT OUTER JOIN SON R.B=S.B AND R.C=S.C

此式表示关系 R 和 S 属性 B 和 C 执行 F 联接操作，并且是左外联接。结果如图 5.17 的（d）所示。

③R FULL OUTER JOIN S USING（B）

此式表示关系 R 和在属性 B 上执行自然联接，并且是完全外联接，属性 B 只保留一个。如图 5.17 的（e）所示。



图 5.17 关系的联接操作

5．3．5 SQL DML 的数据更新语句

数据理新包括数据插入、删除和修改等操作。

1． 插入操作

插入语句 INSERT 语句。有三种方式，一种是单元组的插和入，一种是多元组的"表"表达式插入方式，第三种是多元组的 SELECT 查询的插入方式。下面举例说胆。

[例 5．25]①往基本表 SC 中插入成绩元组（S4，C6，90），可用下列语句：

INSERT INTO SC（S#，C#，GRADE）

VALUES（'S4'，'C6'，'90'）

②往基本表插入多个元组，在 SQL3 中允许的。例如往 SC 中插入三个元组，可用下列语句：

INSERT INTO SC(S#，C#，GRADE)

VALUES('S4','C4',85),

('S3','C6',90)

('S7','C2',70)

③ 在基本表 SC 中，把平均成绩大于 80 分的男学生的学号和平均成绩存入另一个关系 S_GRADE（S#，AVG_GRADE），可用下列 SELECT 查询的插入方式实现：

```
INSERT INTO S_GRADE（S#，AVG_GRADE）

SELECT S#，AVG（GRADE）

FROM SC

WHERE S# IN

（SELECT S#

FROM S

WHERE SEX='M'

GROUP BY S#

HAVING AVG（GRADE）>80
```

2. 删除操作

SQL 的删除操作是指从基本表中删除元组。删除语句的句法如下：

```
DELETE FROM R

[WHERE F]
```

其语义是：从基本表 R 中删除满足条件 F 的元组。可以看出，该语句实际上是"SELECT * FROM R WHERE F"和 DELETE 动作的结合，每找一个元组，就把它删去。就注意，DELETE 语句只能从个基本表中删除元组，而不能一次从多个表中删除元组。WHERE 子句中条件可以嵌套，也可以来自几个关系的复合条件。

[例 5.26]把 C4 的课程中小于该课程平均成绩从基本表 SC 删去可用下列语句实现：

```
DELETE FROM SC

WHERE C#='C4'

AND GRADE <（SELECT AVG（GRADE）

FROM SC

WHERE C#='C4'
```

3. 修改操作

SQL 中用 UPDATE 语句修改基本表中的属性值。其句法如下：

UPDATE R

SET 列名 =值表达式[，列名=值表达式……]

[WHERE F]

其语义是：修改基本表 R 中满足条件表达式 F 的元组中的属性值，修改的值在 SET 子句中组出。

[例 5．27]①在基本表 SC、C 中，把课程名为 Maths 的成绩提高 10%，可用下列语句实现：

UPDATE SC

SET GRAD=GRADE * 1.1

WHERE C# IN

（SELECT C#

FROM C

WHERE CNAME='Maths'）

③ 当 C4 课的成绩低于该门课程的平均成绩时，提高 5%，可用下列语名实现：

UPDATE SC

SET GRADE=GRADE * 1.05

WHERE C#='C4'

AND GRADE <（SELECT AVG（GRADE）

FROM SC

WHERE C#='C4'

5．3．6 视图定义和对视图的操作

在 SQL 中，外模式一级数据结构的基本单位称为视图。视图采用 SELECT 语句来构造。在定义了一个视图时，只是把其定义存放在系统的数据字典中，直到用户使用视图时才去求对应的数据。

1. 视图的定义

定义视图可用"CREATE VIEW"语句实现。下面举例说明。

[例 5．28]用户经常要在基本表 SC 查询每个学生选修课程（成绩为非空）的门数和平均成绩，可用下列形式定义一个视图 S_GRADE 供用户使用：

```
CREATE VIEW S_GRADE（S#，C_NUM，AVG_GRADE）

AS SELECT S#，COUNT（C#），AVG（GRADE）

FROM SC

WHERE GRADE IS NOT NULL

GROUP BY S#
```

2. 视图的撤销

可以用"DROP VIEW"语句把视图从系统中撤销。例如撤销视图 S_GRADE 可用下列语句：

```
DROP VIEW S_GRADE
```

3. 对视图的更新操作

在视图定义以后，对于视图的查询（SELECT）操作，可像对待基本表一样使用，没有什么限制。但对于视图的更新操作（INSERT，DELETE，UPDATE），有以下三条规则：

（1） 如果一个视图是从多个基本表使用联接操作导出，那么不允许对这个视图执行更新操作。

（2） 如果在导出视图的过程中，使用了分组和聚合操作，那么不允许对这个视图执行更新操作。

（3） 如果视图是从单个基本表使用选择、投影操作导出的，并且包含了基本表的主键或某个候先键，那么这样的视图称为"行列子集视图"，可执行更新操作。在 SQL2 中，允许更新的视图在定义时，必须加上"WITH CHECK OPTION"短语。

4. 视图的优点

视图是用户一级的数据观点，由于有了视图机制，数据库系统具有下列优点：

（1） 视图提供了逻辑数据独立性。使应用程序与数据库的整体逻辑结构、存储结构无关。

（2） 简化了用户观点，用户只需关心视图的结构。

（3） 增加了数据库的安全保护功能。

5． 3.7 SQL 的访问控制

SQL 的访问控制功能主要是指对用户访问基本表的控制，主要有授权语句和回收语句。

1． 授权语句：

格式如下：

GRANT 权限表 ON 表名 TO 用户名表[WITE GRANT OPTION]

该语句把表的使用权限授予指定的若干用户。在有 WITH GRANT OPTION 短语时，被授权的用户还可将获得的权限再转授给其他用户。

权限表中的权限可以有：SELECT（检索），INSERT（插入），DELETE（删除），UPDATE（修改），EXPAND（扩弃），INDEX（建立索引），ALL PRIVILEGES（所有上述权限），DBA（用户被系统拽定为 DBA）等。

2． 回收语句

格式如下：

REVOKE 权限表 ON 表名 FROM 用户名表

把已授给指定用户的在指定表上的使用权限收回。

5．3．8 嵌入式 SQL

1． SQL 运行环境

由于 SQL 是基于关系数据模型的语言，而高级语言基于整数、实数、字符、记录、灵敏组等数据类型，因此两者之间尚有很大差别，称为"有缝隙"。例如 SQL 语句不能直接使用指针、数组等数据结构，而高级语言一般不能直接进行集合操作。为了能在宿主语言的程序中嵌入 SQL 语句，必须对使用方式作某此规定。

嵌入式 SQL 的实现，有两种处理方式；一种是扩弃宿主语言的编译程序，使之能处理 SQL 语句；别一种是采用预处理方式。目前多数系统采用后一种方式。

预处理方式是先用预处理程序对源程序进行扫描，识别出 SQL 语句，并处理成宿主语言过程调用语句；然后再有宿主语言的编译程序编译成目标程序。

2. 嵌入式 SQL 的使用规定

在宿主语言程序中使用 SQL 语句有以一下规定：

（1）在程序中要区分 SQL 语句与宿主语言的语句：所有的 SQL 的语句前必须加上前缀标识"EXEC SQL"并以"END_SQL"作为语句结束标志。即嵌入的 SQL 语句的格式如下：

EXEC SQL <SQL 语句>END_EXEC

结束标志在不同的宿主语言中是不同的，在 C 和 PASCAL 中规定结束标志不用"END_EXEC"，而使用分号"；"。

（2）允许嵌入的 SQL 语句中引用宿主语言的程序变量，但有两条规定：

• 引用时，这些变量（称为"共享变量"）前必须加冒号作为前缀标识，以示与数据库中变量有区别。

• 这些变量由宿主语言的程序定义，并用 SQL 的 DECLARE 语句说明。例如在 C 程序中可有下列形式说明共享变量：

```
EXEC SQL BEGIN DECLARE SECTION;

Char sno[5],name[9];

Char SQLSTATE[6];

EXEC SQL END DECLARE SECTION;
```

上面四行组成一个说明节，说明了三个共享变量，其中 SQLSTATE 是一个特殊的共享变量，起着解释 SQL 语句执行状况的作用。当一个 SQL 语句执行成功时，系统自动给 SQLSTATE 赋上全零值（即"00000"），表示未发生错误；否则值为"非全零"，例如"02000"表示未找到元组。在执行一个 SQL 语句后，程序可以根据 SQLSTATE 的值转向不同的分支，以控制程序的流向。

（3）SQL 的集合处理方式与宿主语言单记录处理方式之间的协调，用游标（CURSOR）技术实现。与游标有关的 SQL 语句有下列四个：

① 游标定义语句（DECLARE）

游标是与某一查询结果相联系的符号名，游标用 SQL 的 DECLARE 语句定义，句法如下：

EXEC SQL DECLARE<游标名>CUPSOR FOR

<SELECT 语句>

END_EXEC

这是一个说明语句，定义中的 SELECT 语句并不立即执行。

② 游标打开语句（OPEN）

该语句执行游标定义中的 SELECT 语句，同时游标处于活动状况。游标是一个指针，此时指向查询结果的第一行之前。OPEN 语句的句法如下：

EXEC SQL OPEN<游标名> END_EXEC

③ 游标推进语句（FETCH）

该语句使用时，游标推进一行，并把游标指向的行（称为当前行）中的值取出，送到共享变量中。其句法如下：

EXEC SQL FETCH FROM <游标名> INTO <变量表> END_EXEC

变量表是由用逗号分开的共享变量组成。该语句常置于宿主语言程序的循环结构中，并借助宿营主语言的处理语句逐一处理查询结果中的一个个元组。

④ 游标关闭语句（CLOSE）

该语句关闭游标，使它不再和查询结果相联系。关闭了游标，可以再次打开，与新的查询结果相联系。该语句的句法如下：

EXEC SQL CLOSE <游标名> END_EXEC

在游标处于活动状态时，可以修改和删除游标指向的元组。

3．嵌入式 SQL 的使用技术

SQL DDL 语句，只要加上前缀标识"EXEC SQL"和结束标志"END_EXEC"就能嵌入在宿主语言的程序中使用。SQL DML 语句嵌入使用时，分成是否使用游标两种方法，下面介绍这两种方法。

（1） 不涉及游标 SQL DML 语句

如果是 INSERT、DELETE 和 UPDATE 语句，那么加上前缀标识"EXEC SQL"的结束标志"END_EXEC"，就能嵌入在宿主语言的程序中使用。对于 SELECT 语句中要增加 INTO 子句，指出找到的值应送到相应的共享变量中去。

[例 5．29]给出的 C 程序中不涉及游标的嵌入式 SQL DML 语句的使用例子。

① 在基本表 S 中根据共享变量 givensno 值检索学生的姓名、年龄和性别，语句的写法如下：

```
EXEC SQL SELECT sname,age,sex

INTO :sn,:sa,ss

FROM s

WHERE s#=:givensno;
```

② 从基本表 SC 中删除一个学生的所有成绩，这个学生的姓名在共享变量 SN 中给出其语句如下：

```
EXEC SQL DELETE FROM sc

WHERE s#= (SELECT S#

FROM S

WHERE sname=: sn);
```

（2） 涉及游标的 SQL DML 语句

当 SELECT 语句查询结果是多个元组时，宿主语言程序无法处理，一定要用游标机制把多个元组一次一个地传送给宿主语言程序处理。

[例 5．30]在基本表 SC 中检索某学生（学号由共享变量 givensno 给出）的学一成绩信息（S#,

C#, GRADE），下面是该查询的一个 C 函数：

```
# define NO_MORE_TURLES!(strcmp(SQLSTATE,"02000"))

void sel()

{EXEC SQL BEGIN DECLARE SECTION;

char sno[5],cno[5],givensno[5];

int g;

char SQLSTATE[6];
```

```
EXEC SQL ENDDECLARE SECTION;

EXEC SQL DECLARE scx CURSOR FOR

SELECT s#,c#,grade

FROM sc

WHERE s#=:givensno;

EXEC SQL FETCH FROM sc

INTO :sno,:con,g;

If (NO MORE TURLES )break;

Printf("%s,%s.%d",sno cno,g);

}

EXEC SQL CLOSE scx;

}
```

这里使用了 C 程序中的宏定义 NO MORE TUPLES，表示找不到元组，其值为 1。

在游标处于活动状况时，可以用 UPDATE 或 DELETE 语句修改或删除游标指向的元组。

4．动态 SQL 语句

前面提到的嵌入式 SQL 语句都必须在源程序中完全确定，然后再由预处理程序编译和宿主语言编译程序编译。在实际使用时，源程序往往不能包括用户的所有操作。用户对数据库的操作有时在系统运行时才能提出来，这时要用到嵌入式 SQL 的动态技术才以实现。动态 SQL 技术主要有两个 SQL 语句：

（1） 动态 SQL 预备语句。句法如下：

EXEC SQL PREPARE〈动态 SQL 语句名〉FROM〈共享变量或字符串〉；

此处共享变量或字符串的值应是一个完整的 SQL 语句。这个 SQL 语句可以在程序运行时由用户输入才组合起来。此时，这个 SQL 语句并不执行。

（2） 动态 SQL 执行语句。句法如下：

EXEC SQL EXECUTE〈动态 SQL 语句名〉；

动态 SQL 语句使用时，还可以两点改进技术：

• 当预备语句组成的 SQL 语句合而只需执行一次，那么预备语句和执行语句要合并成一个语句：
EXEC SQL EXECUTE IMMEDIATE〈共享变量或字符串〉；

• 当预备语句组成的 SQL 语句的条件值尚缺时，可以在执行语句中用 USINGT 短语补上：
EXEC SQL EXECUTE〈 动态 SQL 语句名〉USLING〈共享变量〉；

[例 5．31]下面两个 C 程序段说明了动态 SQL 语句的使用技术。

① EXEC SQL BEGIN DECLARE SECTION;

Char * query;

EXEC SQL END DECCLARE SECTION;

Scanf ("%s",query); / * 从键盘输入 一个 SQL 语句到字符数组中，字符指针 guery 指向字符串中的第 1 个字符。

如果执行语句只做一次，那么最后两个语句可合并成一个语句：

EXEC SQL EXECUTE IMMEDIATE：query;

② char * query="UPDATE sc

SET grade=grade * 1.1

WHERE C# =?";

EXEC SQL PREPARE dynprog FROM :query;

Char cno[5]="C4";

EXEC SQL EXECUTE dynprog USING :cno;

此处第一个 char 语句表示用户组合成一个 SQL 语句，但有一个值（C#）还不能确定，因此用"？"表示。第二个语句是动态 SQL 预备语句。第三个语句（char 语句）表示取到 C#值。第四个语句是动态 SQL 执行语句，"？"值到共享变量 cno 中取。

5.4 关系数据库规范化理论

在关系模型中,一个数据库模式是关系模式的集合。关系数据库的规范化理论(即"模式设计理论"),主要研究如何从多种可能的组合中选取一个合适的、性能好的关系模式的集合作为数据库模式,这是数据库设计技术中的一个重要课题。本节介绍函数依赖、模式分解特性、范式等基本概念。

5．4．1 关系模式的存储异常问题

设有一个关系模式 R（SNAME，CNAME，TNAME TADDRESS），其属性分别表示学生姓名、选修的课程名、任课教师姓名和任课教师地址。仔细分析，这个模式存在着下列存储异常的问题：

（1） 数据冗长：如果某门课程有 100 个学生选修，那么在 R 的关系中就要出现 100 个元组，这门课程的任课教师姓名和地址也随之重复出现 100 次。

（2） 修改异常：由于上述冗余问题，当需要修改这个教师的地址时，就要修改 100 个元组中的地址值，否则就会出现地址值不一致的现象。

（3） 插入异常：如果不知道听课学生名单，这个教师的任课情况和家庭地址住处就无法进入数据库；否则就要在学生姓名处插入空值。

（4） 删除异常：如果某门课程的任课教师要更改，那么原来任课教师的地址住处将随之丢失。

因此，模式 R 虽然只有四个属性，但却是性能很差的模式。如果把 R 分解成下列两个关系模式：

R1（SNAME，CNAME）和 R2（CNAME，TNAME，TADDRESS）

则能消除上述提出的存储异常现象。

产生这种异常的根源何在？与关系模式属性值之间的联系直接有磁。在模式 R 中,学生与课程有直接联系,教师与课程有直接联系,而教师与学生无直接联系,这就产生了模式 R 的存储异常。因此,模式设计强调"每个联系单独表达"是一条重要的设计原则,把 R 分解成 R1 和 R2 是符合这条原则的。

关系数据库规范化理论主要包括三方面的内容：

（1） 数据依赖：指数据之间存在的各种联系和约束，例如键（key）就是一种依赖。函数依赖是最基本的一种依赖。

（2） 范式：模式分解的标准形式。

（3） 模式设计方法：设计规范的数据库模式的方法。

5．4．2 函数依赖

数据依赖是现实世界中忏悔间联系各约束的抽象，是数据的内在性质。函数依赖（functional dependency,FD）是一种最重要、最基本的数据依赖。下面先介绍其定义，再介绍推理规则。

1． 函数依赖的定义

没有关系模式 R（U），X 和 Y 是属性集 U 的子集，FD 是开成 XY 的一个命题，只要 r 是 R 的关系，对 r 任意两个元组都是"X 值相等蕴涵 Y 值相等"，那么 FD X→Y 在关系模式 R(U)中成立。

"X→Y"读作"X 函数决定 Y"或"Y 函数依赖于 X"。

X→Y 反映了属性集 X 和 Y 之间的联系，对于每一个 X 值，只有惟一的 Y 值与之对应。

[例 5．32]关系模式 R（S#，C#）→GRADE，C#→TNAME，TNAME→TADDR 等。

2． FD 和键的联系

从 FD 的定义可以看出，如果 FD X→U 的关系模式 R（U）上成立，并且不存在 X 的任一真子集 X 使 X→U 成立，那么称 X 是 R 的一个候选键。也就是 X 值惟一决定关系中的元组。由此可见，FD 是键概念的推广，键是一种特殊的 FD。

[例 5．33]例 5。32 中，对于模式 R 有下列 FD 成立；

（S#，C#）→（S#，C#，GRADE ，TNAME ，TADDR）

因此（S#，C#）是 R 的键，它函数决定 R 中所有属性。

3． FD 推理规则

由于 FD 是命题，因此 FD 之间可以相互逻辑蕴涵，即可以相互推导。1974 年 W。W。Armstrong 提出了 FD 的推理规则.

设 U 是关系模式 R 的属性集,F 是 R 上成立的只涉及到 U 中属性的 FD 集,则有以下三条推理规则

(1) A1(自反性):若 Y$\subseteq$ X$\subseteq$ U,则 X→Y 在 R 上成立.

(2) A2(增广性):若 X→Y 在 R 上成立,且 Z$\subseteq$ U,则 XZ→YZ 在 R 上成立.

(3) A3(传 递性):若 X→Y 和 Y→Z 在 R 上成立,则 X→Z 在 R 上成立.

这里 XZ,YZ 等写法表示 X∪Z,Y∪Z.已经证明,上述三条推理规则是 FD 的一个正确的和完备的推理系统.根据上述三条规则还可以很方便地推出其他三条常用的推理规则:

(1) A4(并规则):若 X→Y 和 X→Z 在 R 上成立,则 X→YZ 在 R 上成立.

(2) A5(分解规则):若 X→Y 在 R 上成立,且 Z⊆ Y,则 X→Z 在 R 上成立.

(3) A6(伪传递规则):若 X→Y 和 WY→Z 在 R 上成立,则 WX→Z 在 R 上成立.

5.4.3 关系模式的分解特性.

1. 模式分解问题

在 5.4.1 节中已提出,关系模式 R 分解成数据库模式 δ={R1, …… RK}的目的是为了消除数据冗余和操作异常现象。设 R 对应的关系为 r， δ 对应的实例为 δ=<r1,……rk>, 那么我们要问， δ 和 r 是否表示同一个数据库？这要从两个角度考虑：

（1） r 和 δ 是否等价，即是否表示同样的数据。这个分解问题是"无损联接"特性表示。

（2） 在关系模式 R 上有一个 FD 集 F，在数据库模式 δ 的每个模式 Ri 上有一个 FD 集 Fi，那么 F 与{F1，……，Fk}是否等价。这个分解问题用"保持依赖"特性表示。

3. 无损联接

[例 5. 34]设有关系模式 R（ABC），分解成 δ={AB，AC}。

（1） 图 5.18 的（a）是 R 上的一个关系 r,(b)和(c)是 r 在 AB 和 AC 上的投影 r1 和 r2，显然，此时满足 r1⋈ r2=r。也就是 r 在投影、联接以后未丢失失信息，这正是我们所希望的。这种分解称为"无损联接分解"。

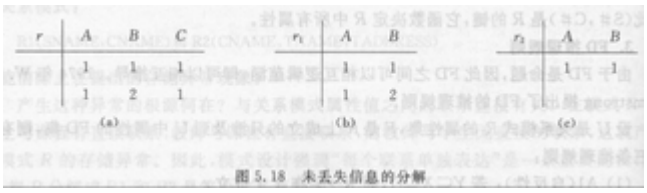
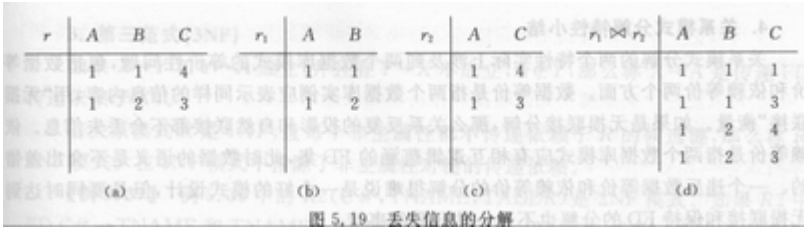


图 5.18 未丢失信息的分解

（2） 图 5.19 的（a）是 R 上的一个关系 r，（b）和(c)是 r 在 AB 和 AC 上的投影 r1 和 r2，（d）是 r1⋈ r2,显然 r1⋈ r2≠r 也就是 r 在投影、联接以后比

原来的元组还要多.但把原来的信息丢失了.这种分解是我们不希望产生的.这种分解称为"损失联接分解".



r	A	B	C
	1	1	4
	1	2	3

(a)

r_1	A	B
	1	1
	1	2

(b)

r_2	A	C
	1	4
	1	3

(c)

$r_1 \bowtie r_2$	A	B	C
	1	1	1
	1	1	3
	1	2	4
	1	2	3

(d)

图 5.19 丢失信息的分解

无损联接分解的开式定下如下:设 R 是一个关系模式,F 是 R 上的一个 FD 集.R 分解成数据库模式 δ={R1,……, RK}。如果对 R 中每一个满足 F 的关系 r 都有下式成立：

$r=\Pi R1(r) \bowtie \Pi R2(r) \bowtie \ldots \bowtie \Pi Rk(r)$

那么称分解 δ 相对于 F 是"无损联接分解"，否则称为"损失联接分解"。其中 ΠRi(r)表示关系 Ri 在模式的属性上的投影。

3. 保持 FD 的分解

设数据库模式 δ={R1……, RK}是关系模式 R 的一个分解，F 是 R 上的 FD 集，δ 中每个模式 Ri 上的 FD 集是 Fi。如果{F1，F2，……，Fk}与 F 是等价的（即相互逻辑蕴涵），那么我们称分解 δ 保持 FD。

如果分解不能保持 FD，那么 δ 的实例上的值就可能有违反 FD 的现象。下例说明了这个问题。

[例 5. 35]设关系模式 R（WNO，WS，WG）的属性分别表示职工的工号、工资级别、工资数目。

FD 是 WNO→WS，WS→WG。R 分解成 δ 把 WS→WG 丢失了，即 δ 不保持 FD.图 5。20 的（a）和（b）是两个关系 r1 和 r2,r1 和 r2,(c)是 $r1 \bowtie r2$。r1,r2 分别满足 F1 和 F2，但 $r1 \bowtie r2$ 违反了 F 中的 WS→WG。



WNO	WS
w1	8 级
w2	6 级
w3	6 级

(a) 关系 r_1

WNO	WG
w1	200
w2	160
w3	140

(b) 关系 r_2

WNO	WS	WG
w1	8 级	200
w2	6 级	160
w3	6 级	140

(c) $r_1 \bowtie r_2$

图 5.20 丢失 FD 的分解例子

4. 关系模式分解特性小结

关系模式分解的两个特性实际上涉及到两个数据库模式的等价性问题,包括数据等价和依赖等价两个方面。数据等价值是指两个数据库实例应表示同样的信息内容，用"无损联接"衡量。如果是无损联接分解，那么关系反复的投影和自然联接都不会丢失信息.依赖等价是指两个数据库模式应有相互逻辑蕴涵的 FD 集,此进数据的语义是不会出差错的.一个违反数据等价和依赖等价的分解很难说是一个好的模式设计,便是要同时达到无损联接和保持 FD 的分解也不是一件容易的事情.

5.4.4 关系模式的范式

关系模式的优劣,用什么标准衡量?这个标准就是模式的范本(normal forms,NF).范式有许多种,与数据依赖有着直接的联系.

1. 第一范式(1NF)

如果关系模式 R 的每个关系 r 的属性值都是不可分的原子值，那么称 R 是第一范式（1NE）的模式， r 是规范化的关系。关系数据库研究的关系都是规范化的关系。

例如教师模式 R（NEME，ADDRESS，PHONE），如果一个教师家里装了两部电话，那么在关系中至少要出现两个元组，以便存储这两个号码。

1NE 的模式是关系数据库最基本的要求，远不是理想的模式。下面根据不同的 FD 提出若干范式。

3. 第二范式（2NE）

我们把关系模式 R 的属性分为两类：一类是键的属性，称为主属性；另一类是不属于任何键的属性，称为非主属性。

我们把 FD 分成完全 FD 和局部 FD 两类。如果 X→Y 成立，对 X 的任何真子集 X 都有 X'→Y 不成立，则称 X→Y 是完全 FD，否则称 X→Y 是局部 FD。

若关系模式 R 是 1NE，且每个非主属性完全函数依赖于候选键，那么称 R 是 2NE 模式。

[例 5. 36]关系模式 R（S#，C#，GRADE，TNAME，TADDR）的属性表示学生学号、选修课程的编号、成绩、任课教师姓名和教师地址等。R 上有 FD（S#,C#）→TNAME 和 C#→TNAME，显然，前一个 FD 是局部依赖，因此 R 不是 2NE 模式。此时 R 的关系就出现冗余的异常现象，譬如某一门课程有 100 个学生选修，那么在关系中就会存在 100 个元组，因而教师的姓名和地址就会重复 100 次。
如果把 R 分解成 R1（S#，C#，GRADE）和 R2（C#，TNAME，TADDR）后，（S#，C#）

→TNAME 在 R1 和 R2 中不出现了，那么刚才的冗余和异常就会消失。R1 和 R2 都是 2NE 模式。

3. 第三范式（3NE）
如果 X→Y 和 Y→A 成立，并且有 Y→X 不成立，A∉Y，那么称 X→A 是传递 FD（A 传递依赖于 X）。
若关系模式 R 是 1NF，且每个非主属性都不传递依赖于 R 的候选键，那么称 R 是 3NE 模式。在 3NE 模式中排除了非主属性对键的传递依赖。

[例 5．37]例 5。36 中的 R2（C#，TNAME，TADDR）是 2NE 模式。如果 R2 中有 FD C#→TNAME 和 TNAME→TADDR 就是一个传递依赖，从而 R2 不是 3NF 模式。在 R2 的关系双会出现新冗余和异常现象，譬如一个教师开设五门课程，那么关系中就会出现五个元组，教师的地址就会重复五次。

如果把 R2 分解成 R21（C#，TNAME）和 R22（TNAME，TADDR）后，C#→TADDR 在 R21 和 R22 中都不出现。这样 R21 和 R22 都是 3NE 模式。例 5。36 中的 R1（S#，C#，GRADE）既是 2ME 也是 3NE 模式。

可以证明，3NE 的模式必定是 2NE 的模式。局部依赖和传递依赖是产生冗余和异常的两个重要原因。由于 3NE 模式中不存在非主属性对候选键的局部依赖和传递依赖，因此消除了很大一部分存储异常，具有较好的性能。而对于非 3NE 的 1NE，2NE，甚至非 1NE 的关系模式，由于它们性能上的弱点，一般不宜作为数据库模式，通常需要将它们变成 3NE 或更高级的范式，这种变换过程，称为"关系的规范化处理"。

4．BC 范式（Boyce_codd NF,BCNF）

若关系模式 R 是 1NE，且每个属性都不传递依赖于 R 的候选键，那么称 R 是 BCNF 模式。此进排除了任何属性对键的传递依赖。

[例 5．38]设 R（B#，BNAME，AUTHOR）的属性分虽表示书号、书名和作者名。如果约定，每个书号只有一个书名，但不同书号可以有相同书名；每本书可以有多个作者合定写，便每个作者参与编写书的书名应该互不相同。这样的约定可以用下列两个 FD 表示：

B#→BNAME 和（AUTHOR，BNAME）→B#

R 的键为（AUTHOR，BNAME）或（B#，AUTHOR），因而模式 R 的属性都是主属性，R 是 3NE 模式。但显然 BNAME 传递依赖于候选键（AUTHOR，BNAME），因此 R 不是 BCNF 模式。在一本书由多个作者编写进，其书名与书号间联系在关系中将多次重复出现，带来冗余和操作异常现象。如果把 R 分解成 R1（B#，AUTHOR），那么能解决上述问题，且 R1 和 R2 都是 BCNF 模式。

上述四种范式之间有如下联系：BCNF⊂3NF⊂2NF⊂1NF

5. 模式分解方法

有关模式分解方法有两个结论：

（1） 可以把一个关系模式无损联接地分解成 BCNF 模式集，便不一定能达到保持 FD 的目标。

（2） 可以把一个关系模式无损联接且保持函数依赖地分解成 3NE 模式集。

至此，我们对关系模式的分解有了了解。关系模式 R 相对于函数依赖集 F 分解成数据库模式 δ={R1，……，RK}，一般应具有下面三项特性：

（1） BNCF 模式集，或 3NF 模式集；

（2） 无损联接，即对于 R 上任何满足 F 的关系 r 应满足 $r=\bowtie_{i=1}^{k}\Pi_{Ri}(r)$

（3） 保持函数依赖，即 δ 中的 FD 集{F1，……，F2}与 R 上的 F 等价（相互逻辑蕴涵）。

数据库设计者在进行关系模式设计时，应作权衡，尽可能使数据库模式保持最优的特性。一般，尽可能设计成 BCNF 模式集。如果设计成 BCNF 模式集的同时达不到保持 FD 的目标。那么只能降低要求，设计成 3NE 模式集，而同时又达到无损联接和保持 FD 的目标。

5.5 数据库的保护

在 DBS 的运行时，DBMS 要对数据库进行监控，以保证整个系统的正常运转，保证数据库中的数据安全可靠、正确有效，防止各种错误的产生，这就是对数据库的保护，有时也称为"数据控制"。对数据库的保护，有进也称为"数据控制"。对数据库的保护能过四个方面实现：数据库的恢复、完整性控制、并发控制和安全性控制。其中每一方面构成了 DBMS 的一个子系统，下面分别介绍。

5．5．1 数据库的恢复

1. 数据库故障和数据库恢复（recovery）

在 DBS 运行过程中，有可能出现各式各样的故障，其中数据库的故障有两类：

第一类故障是数据库已被破坏，根本无法读取数据，或者数据库中大部分数据都有错误，此时数据库已不能用，只能报废。

第二类故障是数据库未被破坏，但某些数据有错误，此时数据库还能用，但必须改正其中的错误数据。造成数据库故障的原因很多，例如：应用程序、系统软件中的错误，设备、通道或 CPU 等硬件的错误，操作员的错误（启动了错误的磁带），电源不稳定，计算机病毒，计算机犯罪等等。

作为 DBMS，应能把数据库从被破坏或不正确的状态，恢复到最近的一个正确状态，这种能力称为"数据库的可恢复性"。恢复工作由 DBMS 的恢复子系统执行。

2. 恢复的基本原则和实现方法

数据库恢复的基本原则就是"冗余"两个字。这里的冗余是指数据库的重复存储。具体实现方法如下：

（1） 平时，经常性地做好下面两件工作

① 周期性地（比如一天一次）对整数据库进行拷贝，或转储到磁带一类的存储介质中。一般的 2～3 份，重要的 5 ～ 6 份，特别重要的数据库通过网络通信转储到远地的"数据银行"。

② 建立"日志"数据库。对数据库的每次更新，记下变更前后的值，写到"日志"中，以备查阅。

（２）如果数据库出现故障，则区别情况分别对待

① 数据库本身破坏了（磁盘坏了）。 此时把最近一次拷贝的数据库装入到新的磁盘，然后借助"日志"，执行"重做"（redo）操作，将两个数据库状态之间的所有修改重做一遍。

② 数据库未被破坏，但某些数据不可靠受到怀疑。例如程序修改数据库时系统异常中断，这时不必去拷贝存档的数据库，只要通过日志数据库执行"撤销"（undo）操作，撤销所有不可靠的修改，把数据恢复到正确的状态。

３． 事务(transaction)

ＤＢＳ运行的基本工作单位是事务。事务相当于操作系统中的进程，一个事务由应有程序中的一组操作序列组成。事务通常以 BEGIN TRANSACTION（事务开始）语句开始，以 COMMIT 或 ROLLBACK 语句结束。COMMIT 称为"事务提交语句"，表示事务执行成功地结束。引进通知系统，数据库已进入一个新的正确状态，该事务对数据库的所有更新均已写回磁盘上的物理数据库中 ROLLBACK 称为"事务回退语句"，表示事务执行不成功地结束。此进通知系统，已发生某种故障，事务不能继续执行。该事务对数据库的更新必须被撤销（即 undo）操作，数据库应恢复到本事务初始状态，然后系统做上述回退动作 。

从终端用户来看，事务是一个原子，是不可分割的操作序列。事务中包括所有操作要么都有做，要么都不做（就效果而言）。事务不应该丢失，或被分割地完成。

４． 事务故障的种类

有了事务概念以后，数据库的故障呆用事务的故障表示。故障可分为以下四类：

（1） 事务本身可预料的故障：在程序中可预先设置 ROLLBACK 语句结束事务。

（2） 事务本身不可预料的故障：这种故障是指事务没有预定的结束点（COMMIN 或 ROLLBACK）结束而引起的故障，例如算术溢出、违反存储保护等。此时由 DBMS 的恢复子系统通过日志，撤消事务对数据为做的改变，回退到事务初始状态。因此，一个事务是一个工作单位，也是一个恢复单位。

（3） 系统故障：引起系统停止运转随之要求系统重新启动的事件称为"系统故障"。如 OS 或 DBMS 代码错误、操作员操作失误、特定类型的硬件错误（CPU 故障）、突然停电等造成系统停目运行，致使事务在执行过程中以非控方式终止。这时内存中的信息丢失，而存储在外存上的数据末受影响。

在系统停止运行后，需要重新启动，让所有非正常终目的事务做回退工作，使数据为恢复到正常状态。在重新启动时，如何确定哪些事务应做恢复工作？一般采用检查点（checkpoint）方法。每隔一段时间（比如虎添翼分钟），系统产生一个检查点，记录下哪些事务在活动。当系统出故障后，从最近的一个检查点记录获得信息，决定哪些事务需要恢复到初始状态，哪些事务需要重做。

（4） 介质故障：主要是指外存被破坏了，此时需拷贝最近的务份装入新的磁盘，然后使用日志重做事务。

5． SQL 语言中的恢复操作

在 SQL 中没有专门的事务开始语句，每一个对数据据为或模式的查询与操作都包含着一个事务的开始。事务的结束语句是 COMMIT（提交）七 ROLLBACK（回退），其意义与前面叙述的一样，例如在 C 程序中书写时的格式为：

```
EXEC SQL COMMIT;

EXEC SQL ROLLBACK;
```

5．5．2 数据为的完整性

1．DBMS 的完整性了系统

数据的完整性（integrity）是指数据为中数据应始终保持正确的状态，防止不符合语义的错误数据的输入和输出，即所谓的"垃圾进垃圾出"所造成的无效操作和错误结果。完整性和准确性、正确性、有效性的意义是一样的。

完整性由 DBMS 的完整性子系统实现，其主要功能有两点：

（1） 监督事务（特别是更新事务）的执行，并测试是否违反完整性。

（2） 如有违反现象，则采取恰当的动作（例如拒绝操作、报告违反情况、改正错误等处理方式）。

完整性子系统是根据"完整性规则集"工作的。完整性规则集是由 DBA 和程序员事先向完整性子系统提供的有关数据约束的一组规则。每一个规则由三部分组成：

（1） 什么时候使用规则进行检查（称为规则的"触发条件"）；

（2） 要检查什么样的错误（称为"约束条件"或"谓词"）

（3） 如果查出错误应该怎么办（称为"ELSE 子句"，即违反时要做的事情）。长 SQL2 中，表达完整性约束的规则有很多形式，例如主键约束、引用完整性规则、属性值的约束、断言等.下面分别介绍.

2. 主键约束

主键是数据库中最重要的一种约束,在基本表中,主键值不允许空,也不允许重复出现.在例 5.15 中,基本表 S 的定义中用主键子句"PRIMARY KEY(S#)"定义主键.

3. 引用完整必规则和外键的约束

在例 5.15 中,基本表 SC 的定义中用外键子句定义外键:

FORELGN KEY (S#) REFERENCES S(S#)

表示 S#在基本表 S 中是主键,在 SC 中是外键.根据引用完整性规则,SC 中的 S#R 值或者为空值,或者是 S 中的 S#值.应注意,外键值是否允许空值,应视实际情况而定.在关系 SC 中,(S#,C#)是主键,因此不允许 S#值出现空值.在实际系统中,作为主键的关系称为"参照关系",作为外键的关系称为"依赖关系".在该例中,S 是基本关系,SC 是依赖关系.在 VISUAL Foxpro 中,这两个表称为父表和子表.在 PowerBuilder 中称为主表和副表.

(1) 删除参照关系的元组时的考虑
删除参照关系的某个元组(即要删除一个主键值)会对依赖关系产生影响.SQL 中规定,系统可能采取的做法有三种:

• RESTRICT 方式:只有当领带关系中没有一个外键值与要删除的参照关系的元组的主键值相对应时,系统才执行删除,否则拒绝此删除操作.

• CASCADE 方式:将依赖关系中所有外键值与参照关系中要删除元组主键值相对应的元组一起删除.

• SET NULL 方式:删除参照关系的元组,并将领带关系中所有与参照关系中被删元组主键值相对应的元组主键相对应的外键值置为空值.

这三种方式,选择哪一种,要视应用环境的语义而定.外键子句可用下列形式定义:

FOREIGN KEY(S#)REFERENCES S(S#)

[ON DELETE [CASCADE | SET NULL]]

如果 ON 短语不写,表示 RESTRICT 方式;选择 CASECADE 或 SET NULL,表示 CASCADE(连锁删除)或 SET NULL(置为空值)方式.

(2) 修改参数关系中主键值勤时的考虑

此情况与(1)类似,不再叙述.在外键子句中可再加一个"ON UPDATE"短语:

FOREIGN KEY（S#）REFERENCES S（S#）

[ON UPDATE [CASCADE | SET NULL]]

3. 在属性值上的约束

（1） 非空值约束

如果我们要求某个属性的值不充许空值，那么可在属性定义后加上关键字"NOT NULL"。

（2） 检查子句（CHECK 子句）

这个子句在例 5。15 中也已用过，并有详细介绍。

（3） 域约束子句

在用"CREATE DOMAIN"语句定义新的域时，也可出现 CHECK 子句，例如：

CREATE DOMAIN SEX CHAR（1）

CHECK （VALUE IN （' F' , ' M' ））；

CREATE DOMAIN AGE SMALLINT

CHECK ((VALUE>=15)AND (VALUE <=25));

5.断言

有时一个完整性约束牵涉较广,与其他基本表、完整性约束或规则有关，那么 SQL2 提供"断言"（assertions）机制让用户编写完整性规则。断言的句法如下：

```
CREATE ASSERTION ASSE1 CHECK
 (NOT EXISTS
 (SELECT *
FROM SC
WHERE C# IN
 ( SELECT C#
FROM C
WHERE TEACHER='Wu'
AND S# IN
(SELECT S#
FROM S
WHIERE S
WHERE SEX='M')
)
);
```

5.5.3 数据库的并发操作

在多用户共享系统中，许多事务可能同时对同一数据进行操作，称为"并发操作"（concurrency），此时ＤＢＭＳ的并发控制子系统负责协调并发事务的执行，保证数据库的完整性不受破坏，同时避免用户得到不正确的数据。

１． 数据库的并发操作带来的问题

（１） 丢失更新问题

（２） 不一致分析问题（读过时的数据）

【例５.４０】在图５.21中，事务Ｔ１和Ｔ２同时对Ａ的值进行操作，并发执行结果是在时间t7丢失了事务Ｔ１对数据库的修改操作，因此这个并发操作是不正确的。

时间	更新事务 T1	数据库中 A 的值	更新事务 T2
t0		100	
t1	FIND A		
t2			FIND A
t3	A := A −30		
t4			A := A * 2
t5	UPD A		
t6		70	UPD A
t7		200	

（FIND 表示从数据库中读值，UPD 表示把值写回到数据库）

图 5.21 在时间 t7 丢失了事务 T1 的更新

【例 5．41】 在图 5．22 中，事务 T 1 对 A 的值减 30，而事务 T 2 只要读 A 的值。但在 t5 时刻， T 1 修改了 A 的值，此时 T 2 使用的 A 值仍是旧值（100），因此造成了不一致。这个问题称为不一致分析问题，即读了过时的数据。

时间	更新事务 T1	数据库中 A 的值	更新事务 T2
t0		100	
t1	FIND A		
t2			FIND A
t3	A := A −30		
t4	UPD A		
t5		70	

图 5.22 在 t5 时刻，事务 T2 仍认为 A 的值是 100

（3）依赖于未提交更新的问题（读了"脏"数据）

【例 5．42】 在图 5．23 中，事务 T1 把 A 的值修改为 90，但尚提交（即未做 COMMIT 操作），事务 T2 紧跟着把未提交的值（90）读出来。随后，事务 T1 做 ROLLBACK 操作，把 A 的值恢复为 100。而事务 T2 仍在使用被撤消了的 A 值 90。在数据库技术中，把未提交的随后又被撤消的数据称为"脏"数据。

时间	更新事务 T1	数据库中 A 的值	读事务 T2
t0		100	
t1	FIND A		
t2	A := A −10		
t3	UPD A		
t4		90	FIND A
t5	* ROLLBACK *		
t6		100	

图 5.23 事务 T2 依赖于事务 T1 的未完成更新（在时间 t4）

这三个问题需要 DBMS 的并发控制子系统来解决。

２． 并发控制方法

处理并发控制的主要方法是采用封锁技术。有两种封：X 封锁和 S 封锁。

（1）排他型封锁（exclusive lock,简称 X 封锁）

X 封锁的含义是：如果事务 T 对数据 A（可以是数据项、记录、数据集以至整个数据库）实现了 X 封锁，那么只允许事务 T 读取和修改数据 A，其他事务要等事务 T 解除 X 封锁以后，才能对数据 A 实现任何类型的封锁。
可见 X 封锁只允许一个事务独锁某个数据，具有排他性。

（2） 共享型封（sjared lock,简称 S 封锁）

X 封锁只允许一个事务独锁和使用数据，要求太严。需要适当入宽，例如可以允许并发读，但不允许修改，这就产生了 S 封锁概念。

S 封锁的含义是：如果事务 T 对数据 A 实现了 S 封锁，那么允许事务 T 读取数据 A，但不能修改数据 A 的所有 S 封锁解除之前决不允许任何事务对数据 A 实现 X 封锁。

３． SQL2 中的并发控制方法

（1）事务的读写类型

SQL 中允许事务有两种类型：

• READ ONLY（只读型）：事务对数据库的操作只能是读操作。在程序中可加下列语句，表示随后的事务均是只读型：

```
EXEC SQL SET TRANSACTION READ ONLY;
```

• READ WRITE （读写型）：事务对数据库的操作可以是读，也可以是写。在程序中可加下列语句，表示随后的事务均是读写型：

```
EXEC SQL SET TRANSACTION READ WRITE;
```

（2）事务的一致性级别
在封锁技术方面，SQL2 提供事务的四种一致性级别让用户选择。这四个级别从高到低如下所述：

• Serializable(可串行化)：允许事务与其他事务并发执行，但系统应保证其正确执行，不发生错误。

• Repeatable read（可重复读）：只许事务读已提交的数据，而且在两次读同一数据时不允许其他事务修改此数据。

• Read committed(读提交数据)：只许事务读已提交的数据。

• Read uncommitted(可读未提交数据)：允许事务已提交或未提交的数据，也就是允许事务读"脏"数据。

上述四种级别在程序中写法如下：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION LSOLATION LEVEL READ UNCOMMITTED;
```

5．5．4 数据库的安全性

数据为的安全性（security）是指保护数据不被破坏和不被非法窃取。安全性问题是所有计算机系统都有的问题，只是在 DBS 中大量数据集中存放，而且是多用户共享，使安全性问题更为突出。系统安全保护措施是否有效是 DBS 主要的性能标之一。与数据为使用直接有关的安全措施有用户标识和鉴定、存取控制、审计、密码存储、视图的保护和日志监视等方式。

1．用户标识和鉴定

用户标识和鉴定，是系统提供的最外层的安全保护措施。其方法是用户在使用机器前必须以一定的方向系统标褒自己的名字或身份，系统在核实鉴定了用户身份以后才能提供机器的使用权。标识和鉴定的方法有下列几种：

（1）用户代号：在定义外模式，就为每个用户提供了一个用户代号，存放在数据字典中。用户上机时，先键入用户代号，经系统鉴别通过以后才可以进入下一步的核实，否则发出警告，并且拒绝用户使用计算机。

（2）口令（password）：为了进一步核实用户，系统常常要求用户输人口令，为保密起见。用户键入的口令不显示在屏幕上。系统核对口令以鉴别用户身份。

在数据库技术中,对于存取权限的定义称为"授权".在关系数据库中授权方式既有集中的也有分散的管理的,集中管理方式有利于保证数据库的安全性,分散管理方式方便了用户但比较复杂,容易影响数据库的安全.在 5.3.6 节中提到的授权语句(GRANT)和回收语(REVOKE)就是属于存取控制方式.

3.审计(AUDIT)

任何系统的安全性措施不可能是完美无缺的,间谍或者其他犯罪者总是想方设法打破控制.所以,当数据相当敏感,或者对数据的处理极为重要时,就必须进行审计来追踪检查现有情况.审计追踪使用的是非曲直个专用文件或数据库,系统自动将用户对数据所有操作记录在上面,利用审计追踪信息,就能重现导致数据库现有状况的一系列事件,以找出非法存取数据的人.

4. 密码存储和传输

为了更好地保证数据的安全,用密码存储数据,对远程终端信息用密码传输防止中途非法截获.在数据输入时,经过加密算法,把数据加密后才予以存储.使用时,由用户调用解密算法,把数据解密后再使用.采用加密处理的方法,安全性高,即使被窃取了存储数据,也难以及时知道正确的数据.不少数据库产品均提供这种数据加密的功能,系统根据用户的要求对数据实现加密或不加密存储.

5. 视图的保护

在 5.3.5 节中已提到视图的机制可以将要保密的数据对无权存取这些数据的用户隐藏起来,这们自动提供了对数据的安全保护.

6. 日志监视

使用日志监视数据库的活动,在日志中记载日期、时间、程序名、地点、存取的数据和存取的类型等。这个日志由 DBA 监督检查，对于可疑的存取，或者对于高度机密数据的存取实施调查。
安全保护的策略还有很多，但要注意，任何安全的措施都是相对的，要根据具体的用户对象、系统对安全的要求和花费的代价来选择恰当的安全措施。

第 6 章 多媒体基础知识

6.1 多媒体的概念和特征

我们从市场上看到,多媒体技术已把电视式的视听消息传播能力与计算机交互控制功能结合在一起，创造出集图、文、声像于一体的新型信息模式，多媒体使计算机业、家电产品以及网络通信等各类信息处理和传播工具的面貌，都发生了显著的变化.

6．1．1 什么是多媒体

"多媒体"一词译自英文"multimedia",而该词又是由 mutiple 和 media 复合而成.媒体(medium)原有两重含义,一是指存储信息的实体,如磁盘、光盘、磁带、半导体存储器等，中文常译为媒质；二是指传递信息的载体，如数字、文字、声

音、图形等，中文译作媒介。与多媒体对应的一词是单媒体（monomedia），从字面上看，多媒体是由单媒体复合而成的。

从语译概念出发，社会上对多媒体定义的一种说法是：″多媒体是以下两种或两种以上媒体组成的结合体：文本、图形、动画、静态视频、动态视频、声音″。也有人据此说，多媒体，凡图文并茂皆为多媒体解释虽然通俗，但却不可取，因为它忽视了今天信息处理中使用的″多媒体″术语已经同人们直观的生活体验之间存在着深层次的区别，同时也没有向人们指出该词内涵的新的知识信息。

目前多媒体大多只利用了人的视觉和听觉，虚拟现实（virtual reality）中用到触觉，而味觉和嗅觉尚未集成进来。对于视觉也主要在可见光部分，其他光谱段也尚未有效地利用。随着技术的进步，多媒体的含义和范围还将会扩展。因此，现在所说的多媒体已不仅仅是现实生活中常规的几种媒体，更主要的是指明一个信息处理的技术领域。

6．1．2 多媒体的关键技术

使计算机具有处理声音、文字、图像等媒体信息的能国是人们向往已久的理想，直到 20 世纪 80 年代末，当人们在数据压缩技术、大规模集成电路（VLSI）制造技术、CD-ROM 大容量光盘存储器，显示技术以及实时多任务操作系统等方面取得突破性进展以后，多媒体技术的发展才成为可能。下面我们来看看支持多媒体的部分关键技术。

1．数据压缩技术

数字化的声音和图像包含大量的数据。例如，一帧 A4 幅面（21.6cm×30cm）的照片,如果用 12 点/mm(dpm)的分辨率采样,每个像素用 24 位彩色信号表示时的数据量是 25MB.而一分钟的声音信号,用 11.02kHz 的采样率,每个采样用 8 位表示时的数据量是约 660KB.如果不经过数据压缩,实时处理数字化的声音和图像信息所需要的存储容量、传输速率和计算速度都是目前的计算机难以承担的。所以说是数据压缩技术的突破打开了多媒体信息进入计算机的大门。

2．大规模集成电路（VISI）制造技术

进行声音和图像信息的压缩处理要求进行大量的计算。有些处理，例如视频图像的压缩还要求实时完成。这样的处理，如果由通用计算机来完成，需要用六七十年代的中型计算机，甚至大型计算机才能胜任。高昂的成本使多媒体技术无法推广。由于 VLSI 技术的进步使得生产价格低廉的数字信号处理器（DSP）芯片成为可能。DSP 芯片是为完成某种特定信号处理设计的，在通用计算机上需要多条指令才能完成的处理，在 DSP 上往往可用一条指令完成，DSP 的价格虽然只有几美元到几百美元，但完成特定处理时的计算能力却很强。例如，如果要以视频即 30 帧/s 的速率，对一幅 256×256 像素分辨率的图像的每个像素作一次运算，所需的计算速度约为 200 万次/s。而作一次 3×3 窗口的卷积运算，则需要进行九次乘法和一次加法,要求每秒完成千万次运算。如采用由 INMOS 公司生产的 A110 芯片，则可在 1/30s 的时间内，在 512×512 像素的图像上，完成 5×5 窗口的卷

积运算而价格还不到一百美元，所以，可以说是 VLSI 技术为多媒体技术的普遍就用创造了必要条件。

3．大容量的光盘存储器（CD-ROM）

数字化的媒体信息虽然经过压缩处理，仍然包含了大量的数据，视频图像在未经压缩处理时的每秒数据量为 28MB，经压缩处理后每分钟的数据量则为 8.4MB.例如,1.2GB 容量的硬磁盘只能存储约 140 多分钟的视频图像.而且硬磁盘存储器的存储介质通常是不可交换的,所以不适宜用于多媒体信息和软件的发行,而大容量只读光盘存储器(CD-ROM)的出现,正好适应了这样的需要.每张 CD-ROM 的外径为 5 英寸,可以存储约 600MB 的数据,并像软磁盘片那样可用于信息交换.大批量生产时价格也相当低廉.

4.实时多任务操作系统

多媒体技术需要同时处理声音、文字、图像等多种媒体信息，其中声音和视频图像还要求实时处理。声音和语间的播放不能中断，视频图像要求视频速率，即 30 帧/ s 更新图像数据,因此需要能支持对多媒体信息进行实时处理的操作系统。以上是与发展多媒体有关的一些主要技术问题，除此以处，还会涉及到另外的技术问题，例如多媒体技术中的标准化问题、多媒体应用软件的制作、多媒体信息的空间组合和时间同步等。

6．1．3 多媒体系统的构成

一般的多媒体系统是由硬件系统、多媒体工作平台、媒体处理系统工具和用户应用软件构成，这四个部分的具体内容是：

（1）多媒体硬件系统：包括计算机硬件、声音/视频处理器、多种媒体输入/输出设备及信号转换装置、通信与设备及接口装置等。其中最重要的是根据多媒体技术标准而研制生产的多媒体信息处理芯片和相应的板卡以及光盘驱动器等外设。例如：

- IC 类：音频/视频芯片组

  视频压缩/还原芯片组

  数字/模拟转换芯片

  数字信号处理芯片（DSP）

  网络接口芯片

  图形图像控制芯片

- 板卡类：音频处理卡（声卡）

视频处理卡、视频采集/播放卡

图形显示卡、图形加速卡

VGA/TV 转换卡

视频压缩/解码卡（MPEG 卡、JPEG 卡）

有关外设接口总片和板卡

网络通信有关的板卡（如网卡）

• 外设类：摄像机/录放像机

数字照像机/头盔显示器

扫描仪

激光打印机

液晶显示器/高分辨率显示器

光盘驱动器/光盘盘片制作机（如刻录机）

光笔/鼠标/传感器/触摸屏

麦克风/喇叭（音箱）

传真机（FAX）

可视电话机

• MPC（多媒体个人电脑）规范所指定的配置模式

（2） 多媒体工作平台：或称为多媒体核心系统（ multimedia kernel system），具有实时任务高度、多媒体数据转换和同步控制、对多媒体设备的驱动和控制以及图形用户界面管理等。一般是在已操作系统基础上扩充和改造，或者重新设计。例如：

• Intel/IMA 在 DVI（数字视频交互）系统开发中推出的 AVSS（音频视频子系统）和 AVK（音频视频核心系统）。

• APPLE 公司在 Macintosh 上推出 System7.0 中提供 Quick Time 多媒体操纵平台

·Microsoft 公司在 PC 上推出的 Windows With Multimedia Extension1.0 或 Multimedia Development Kit .

(3) 媒体处理系统工具：或称为多媒体系统开发工具软件，是多媒体系统的重要组成部分。从系统工具的功能角度划分，包括以下几类：

·媒体创作软件工具用于建立媒体模型，产生媒体数据。如：Macromedia 公司的 Extreme 3D ,为三维图形视觉空间的设计和创作，提供了包括建模、动画、渲染以及后期制作等功能，直至专业级视频制作；Autodesk 公司的 2D Animation 和 3D Studio 等也是受欢迎的媒体创作工具。用于 MIDI 文件（数字化音乐乐器接口标准)处理的软件也很多，如 Music Time、Recording Session 、Master Track Pro、Studio for windows 等。波形声音工具，如 MDK 中的 Wave Edit、Wave Studio 等。

·多媒体节目写作工具提供不同的编辑、写作方式。如：基于脚本语言帮助创作者控制各种媒体数据的播放，这种写作工具的典型是 Toolbock 其中 OpenScript 语言允许对 Window 的 MCI （媒体控制接口）进行调用，控制各类媒体设备的播放或录制。另一种写作工具是基于流程图的，如 Authorware、Iconauther ，使用流程图来安排节目，每个流程图同许多图标组成，这些图标扮演脚步本命令的角色，并与一个对话框对应，在对话框输入相应内容即可。基于时序的写作工具是能过将元素和事件沿时间轴线安排来达到使多媒体内容演示的同步控制，如 Action 就是这种写作工具。

·媒体播放工具可以直接在电脑上播出，也可以在消费类电子产品中播出。如：Video for Windows，对视频序列（包括伴音）进行一系列处理，可实现软件播放功能；Intel 公司推出 Indeo，在技术上更前进一步，在纯软件视频播放上，提供了功能先进的工具。

·其他各类媒体工具如多媒体数据库管理系统、Video-CD 制作节目工具、基于媒体板卡（如 MPEG 卡）的工具软件、多媒体出版系统工具软件、多媒体 CAL 制作工具等，还有各式 MDK（多媒体平开发平台）等，都非常受欢迎。

（4） 用户应用软件：根据多媒体系统终端用户要求而定制的应用软件，如特定的专业信息管理系统、语音/Fax /数据传输调制管理应用系统、多媒体监控系统、多媒体 CAI 软件、多媒体彩印系统等。在人们探讨应用数字多媒体技术解决自己面临的应用实际问题时，设计建造出各式各样的应用软件系统，使最终用户使用的多媒体系统能够方便、易学、好用，因此，我们把系统用户应用软件视为多媒体系统的必要组成部分。重视多媒体系统应用软件的开发，有利于多媒体技术和系统的普及和推广，更好地发挥其社会效益。

除上述面向终端用户定制的应用软件外，另一类是面向某一领域的用户应用软件系统，这是面向大规模用户的系统产品，如多媒体会议系统、视频点播服务(VOD)等，医用、家用、军用、工业应用等已成为多媒体应用的重要组成方面，多领域应用的特点和需求，推动了多媒体系统用户应用软件的研究和发展。

6．1．4 声卡和显卡

1．声卡

声卡即音频卡，其基本功能是产生与处理声音信号。这些信号在电脑里是以数字的形态表示的，数字信号经声卡芯片处理后，再经由声卡上的数模转换器转换为人耳能识别的模拟信号。如果要处理外部输入的声音信号，声卡还要能过模数转换器先将这些信号转换成电脑能识别的数字信号，再交由声卡芯片处理。

声卡的工作原理是主机能过总线将数字化的声音信号以 PCM（脉码调制） 的方式送到数模（D/A）转换器，将数字信号变成模拟的音频信号。同时又可以通过模数（A/D）转换器将麦克风或 CD 输入的信号转换成数字信号，送到计算机进行各种处理。
图 6.1 是某一声卡的照片其中包含以下一些组成部分：

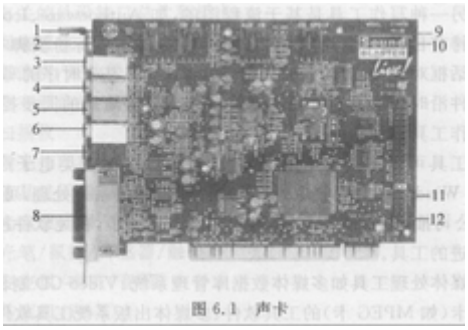


图 6.1 声卡

（1）电话自动应答设备接口（telephone answering device ,TAD）：与 modem 卡上的相应端口相连接，配合软件可使电脑具备电话自动应答功能。

（2） 模拟 CD 音频输入接口（CD-IN）：作用是将来自光驱的模拟音频信号接入声卡，模拟音频线在声卡端的接头一般有两种排列方式，应选用与该接口相匹配的才能确保 CD 音频的正常接入。

（3） 数字输出接口：用于输出数字音频信号。配合声卡上的 AC-3 解码功能，就可输出音效，令观赏 DVD 等影片时更加逼真。

（4） 线输入插孔（LINE IN）：作用是将来自收音机、随身听或电视机等任何外部音频的设备的声音信号输入电脑，可用于录制电视节目伴音、将磁带转达成 MP3 音乐等。

（5） 话筒输入插孔（MIC IN）：可连接适合电脑使用的话筒作为声音输入设备，用于录音、娱乐及语音识别等。如果要打网络电话、用电脑来唱卡拉 OK 时，也少不了它。

（6） 线输出插孔（LINE OUT）：它负责将声卡处理好的声音信号输出到有源音箱、耳机或其他音频放大设备（如功放），这是第一个输出孔，用于连接前端音箱。

（7） 第二个线输出插孔：用于连接后端音箱。四声道以上的声卡都会有两个线输出插孔。

（8） 游戏/MIDI 插口：用于连接游戏杆、手柄、方向盘等外接游戏控制器，也可连接外部 MIDI 乐器（如 MIDI 键盘、电子琴等），配以专用软件要可将电脑作为桌面音乐制作系统使用。

（9） 数字 CD 音频输入接口（CD-SPDIF）：作用是接收来自光驱的数字音频信号，最大限度地减少声音失真。光驱的 Digital Out 接口与声卡上的 CD-SPDIF 输入端连接，可以得到比模拟 CD 音频要更纯净的音质。

（10）辅助音频输入口（AUX-IN）：负责把来自电视卡、DVD 解压卡、MPEG 编/解码卡等设备的声音信号输入声卡。这样就可使各种设备输出的声音信号都能过声卡送至音箱。

（11）声音处理芯片：是整块声卡的核心部分。包括 WAVE 波形的采样与合成、MIDI 音乐的合成以及混音器、效果器的功能都在此芯片内部实现。

（12）扩展功能插针：通过数据线接出，主要用于扩展卡上的输入输出接口。

按照声卡的应用环境，早期的声卡基本可以分为 DOS/GAME 和 Windows 两种环境。这两种声卡分别以"声霸"（Sound Blaster,或 Adilid）和 Windows Sound System 为代表。

从声卡所采用的技术上来看，可分为三类：一是 DSP 技术为基础的声卡，这种声卡本身带有一个 DSP 处理器，所以大部分的控制功能都可以不依赖主机而自行完成，与主机之间只需进行数据交换；二是全硬件声卡，这种声卡采用称为 CODEC（编码与解码）的芯片，控制声音的采样与播放，而其他控制全部依赖于主机，占用较多的主机时间，但成本地很低；三是结合一类和二类两种声卡的优点，采有有阴可编程控制器，使声卡具有一定能力的自管理功能，又不致于成本太高、复杂，它们以硬件实现 DSP 的某些功能，以减轻主机的负担，同时又保持低成本，省去 DSP 总片的开销。

2.显卡

显卡是显示卡的简称。早期的显卡并不受到太多的重视，它的作用只是如实地将处理器输出的数据转换后显示在显示器上。随着图形软件的不断增多，尤其是图形系统的逐渐普及，单单依靠 CPU 去处理所有的数据会极大地影响系统的整体性能，于是出现了加速处理，而 CPU 则可以处理其他重要的任务。图 6.2 是一显卡的照片。

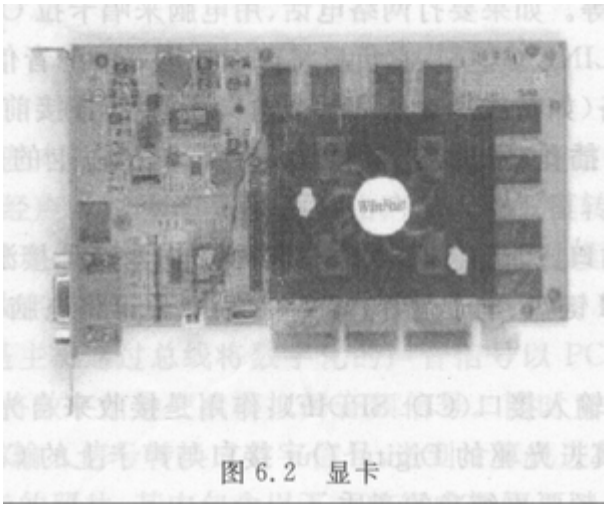


图 6.2 显卡

实际上现在的显卡已都是图形加速卡，都可以执行一些图形处理功能。通常所说的加速卡的性能，是指加速上卡上的芯片集能够提供图形函数计算能力，这个芯片集通常也称为加速器或图形处理器，一般来说在芯片集的内部会有一个时钟发生器、VGA（视频图形陈列）核心和硬件加速功能。

从显示模式的发展历史看，显卡可以分为 MDA 单色显卡、CGA 彩色显卡、EGA 彩色显卡、VGA 彩色显卡等。目前大多是 SVGA 和 XGA。

从显卡所有的总线的历史看，显卡又可以分为 ISA 总线卡、EISA 总线卡、MCA 总线卡、VL 总线卡、PCI 总线卡。目前 PC 大多用 AGP 卡。

作为显卡的重要组成部分，显存也一直随着加速芯片发展而逐步改变着。从早期的 DRAM 到现在广泛泞的 SDRAM 和 SGRAM 等，显存的速度以及它对 3D 加速卡性能的影响也越来越大。显存也被称为帧缓存，通常它是用来存储显示芯片（组）所处理的数据信息。当显示芯片处理完数据后会将数据输送到显存中，然后 RAMDAC 从显存中读取数据并将数字信号转换为模拟信号，最后将信号输出到显示屏。所以，显存的速度、容量以及带宽直接影响着一块加速卡的速度。
下面组出一些显卡的常用术语。

（1）屏幕更新频率（vertical refresh rate）：指显示器每秒能对整个画面重复更新的次数。一般而言，此数值越高，画面就越柔和，眼睛越不会觉得屏幕在闪烁。屏幕更新频率就在 75Hz 以上。

（2）色彩（color depth）：显示画面的色彩数。目前常见的色彩数有 2 bpp(16 色)，8bpp(256 色)，高彩(high color)为 15 bpp(32 768 色)或 16 bpp(65 563 色)，真彩(true color)则为 24bpp(16 777 216 色，即 16M 色)32 bpp（颜色数相同，但可利用 Windows95/98 加速）。Bpp 是指表示每个像素 (pixel)的所需的二进制数的位数。・・・・

（3）分辨率（resolution）：显示画面的细腻程度。一般以画面的最大"水平点数"乘上"垂直点数"来表示。如 800×600，1028×768，1280×1024 等。

（4）DDC（display data channel）:DDC 亦是由 VESA 联盟所制定，这是一种计算机系统与显示器之间的联系信道，主要是让显卡与支持 PnP（即插即用）的操作系统（如 Windows 95/98）相互搭配且互相沟通。DDC 目前分为 DDC1 与 DDC2B，前者是单向信道，只能由主机取得显示器讯息；DDC2B 则是双向信道，能让显卡与显示器之间相互沟通，针对各种不同分辨率做自动化设定。

（5）RAMDAC（RAM digital-to-analo converter）:它是负责将显卡上的数字（ digital）影像数据，转成模拟（ analog）影像讯号输出的芯片。

6.2 图像和图形

人类绘制图画已经有几千年历史。20 世纪 80 年代末多媒体技术出现之后，计算机不仅能处理数字和字符，还能处理声音、图像等综合信息。

6．2．1 基本概念

1.亮度、色调和饱和度

视觉上的彩色可用亮度、色调和饱和度来描述，任一彩色光都是这三个特征的综合效果。

亮度是光作用于人眼时所此起的明亮程度的感觉，它与被观察物体的发光强度有关；由于其强度 不同，看起来可能亮一些或暗一些。对于同一物体照射的光越强，反射光也越强，感觉越亮，对于不同物体在相同照射情况下，反射性越强者看起来越亮。显然，如果彩色光的强度降至使人看不清了，在亮度等级上它应与黑色对应；同样如果其强度变得很大，那么亮度等级应与白色对应。此外，亮度等级上它应与黑色对就；同样，如果其强度变得很大，那么亮度等级就与白色对应。此外，亮度感还与人类视觉系统的视敏功能有关，即使强度相同，颜色不同的光进入视觉系统，也可能会产生不同的亮度。

色调是当人眼看到一种或多种波长的光时所产生的彩色感觉，它反映颜色的种类，是决定颜色的基本特性。如红色、绿色等都是指色调。不透明物体的色调是指该物体在日光照射下，所反射的各光谱成分作用于人眼的综合效果；透明物体的色调则是透过该物体的光谱综合作用的效果。

饱和度是指颜色的纯度，即掺入白光的程度，或者说是指颜色的深浅程度。对于同一色调的彩色光，饱和度越深，颜色越鲜明，或者说越纯。例如，当红色加进白光之后冲淡为粉红色，其基本色调还是红色，但饱和度降低；换句话说，淡色的饱和度比深色要低一些。饱和度还和亮度有关，因为若在饱和的彩色光中增加白光的成分，由于增加了光能，因而变得更亮了，但是它的饱和度却降低了。如果在某色调的彩色光中掺入别的彩色光，会引起色调的变化，掺入白光时仅引起饱和度的变化。

通常把色调通称为色度，上述内容可以总结为：亮度表示某彩色光的明亮程度，而色度则表示颜色的类别与深浅的程度。

2.三原色（RGB）原理

自然界常见的各种颜色光，都可由红（R）、绿（G）、蓝（B）三种颜色光按不同比例相配制而成；同样绝大多数颜色光也可以分解成红、绿、蓝三种色光，这就是色度学中最基本的原理--三原色原理。当然三原色的选择并不是惟一的，也可以选择其他三种颜色为三原色，但是，三种颜色必须是相互独立的，即任何一种颜色都不能由其他两种颜色合成。由于人眼对红、绿、蓝三种色光最敏感因此由这三种颜色相配制所得的彩色范围也最广，所以一般都选用这三种颜色作为基色。

3.彩色空间

（1）RGB 彩色空间：在多媒体计算机技术中，用得最多的是 RGB 彩色空间表示。因为计算机的彩色监视器的输入需要 R、G、B 三个彩色分量，通过三个分量的不同比例，在显示屏幕上可以合成所需要的任意颜色，所以不管多媒体系统采用什么形式的彩色空间表示，最后的输出一定要转成 RGB 彩色空间表示。

（2）YUV 彩色空间：在现代彩色电视系统中，通常采用三管彩色摄像机或彩色 CCD 摄像机，把摄得的彩色图像信号经分色棱镜分成 R_0、G_0、B_0 三个分量的信号；分别经放大和校正得到三基色，再经过矩阵变换电路得到亮度信号 Y、色差信号 R-Y 和 B-Y，最后发送端将 Y、R-Y 和 B-Y 三个信号进行编码，用同一信道发送出去这就是我们常用的 YUV 彩色空间。采用 YUV 彩色空间的优点如下：

①亮度信号 Y 解决了彩色电视机与黑白电视机的兼容问题；

②大量的实验表明，人眼对彩色图像细节的分辨本领比对黑白的低得多，因此对色差信号可以采用"大面积着色原理"，即用亮度信号 Y 传送细节，用色差信号进行大面积涂色。因此彩色图像的清晰度由亮度信号的带宽保证（PAL 制这亮度信号 Y 的带宽采用 4.43MHz），而把色差信号的带宽变窄，PAL 制色差信号宽限制在 1.3MHz。因此一像素可以用更少的位数表示

（3）其他彩色空间表示：彩色窖表示还有很多种如 CIE（国际照明委员会）制定的 CIE XYZ、CIE LAB 彩色空间和 CCIR(Consultative Committee International Radio）制定的 CCIR601-2YC_bC_r 彩色空间等。

6.2.2 图形、图像格式

PC 的处理能力日益增加，加之各种 AGP 显示卡、数码像机的出现，使得大部分原先只能在 Macintosh 机、SUN 或 SGI 图形工作站上完成的图形处理可以直接在ＰＣ上从容进行。又因ＰＣ 的价格比图形工作店低廉，出现了大量的优秀的ＰＣ版本图形处理软件，比如 CorelDraw,Adobe Photoshop,Adobe PhotoStyler,Freehand 等。

图形（图像）格式大致可以分为两大类：一类为位图；另一类为描绘类、矢量类或面向对角的图形（图像）。前者是以点阵即像素形式描述图形（图像）的，后者是以数学方法描述的由几何元素组成的图形（图像）。一般说来，后者对较图像的表达细致、真实，缩入后图形（图像）的分辨率不变，在专业级的图形（图像）处理中运用 较多。

图形（图像）的主要指标为分辨率、色彩数与灰度。对于分辨率一般有屏幕分辨率和输出分辨率两种，前者用每英寸行数与列数表示，数值越大，图形（图像）质量越好；后者衡量输出设备的精度，以每英寸的像素点数表示，数值截止大越好。图形（图像）是 16 位图像，好为 2 的 16 次方，共可表现 65536 种颜色。当图形（图像）达到 24 位时，可表现 1677 万种颜色，即真彩。常见的色彩位表示一般有 2 位、4 位、8 位、16 位、24 位、32 位、64 位这几种。

下面通过图形文件特征后缀名来逐一介绍图形文件格式。

（1）BMP(bit picture):PC 机上最大常见的位图格式，有压缩和不压缩两种形式，它的诞生得益于 Windows OS/2 操作系统。BMP 格式是 Windows 中附件内的绘画小应用程序的默认图形格式，一般 PC 图形（图像）软件都能对其进行访问，以 BMP 格式存储的文件容量较大。该格式可表现从 2 位到 24 位的色彩，分辨率也可从 480×320 至 1024×768。该格式在 Windows 环境下相当稳定，所以在对文件大小没有限制的场合中运用最为广泛。

（2）DIB（device independent bitmap）：描述图像的能力基本与 BMP 相同，并且能运行于多钟硬件平台，只是文件较大。

（3）PCX（PC paintbrush）:由 Zsoft 公司创建的一种经过压缩且节约磁盘空间的 PC 位图格式，它最高可表现 24 位图形（图像）。过去有一定的市场，但随着 JPEG 的兴起，其地位已逐渐降低了。

（4）DIF（drawing interchange format）:Auto CAD 中的图形文件，它以 ASCII 方式储存图形，表现图形在尺寸大小方面十分精确，可以被 CorelDRAW,3DS 等软件调用编辑。

（5）WMF（windows metafile format）:Microsoft Windows 图元文件，具有文件短小、图案造型化的特点，整个图形内容常由各独立组成部分拼接而成。但该类图形比较粗糙，并只能在 Microsoft Office 中调用编辑。

（6）GIF（graphics interschange fromat）: 在各种平台的各种图形处理软件上均可处理的经过压缩的图形格式。它是可以在 Macintosh,Amiga,Atati,IBM 机器间进行移植的标准位图格式。该格式由 Compuserver 公司创建，存储色彩最高只能达到 256 种。由于存在这种限制，除了二维图形软件 AnimatorPro 和 Wed 网页还在使用它外，基他场合已很少使用了。

（7）JPEG（joint photographics expert group）：JPEG 格式可以大幅度地压缩图形文件。同样一幅画面，用 JPEG 格式储存的文件是其他类型图形文件的

1/10-1/20，一般文件大小只有几十 K 或二二百 K，而色彩数最高可达到 24 位，所以它被广泛运用于 internet 上，以节约网络传输资源。同样，为了在一张光盘存储更多的图形（图像），CD 出版商也多采用 JPEG 格式。JPEG 文件之所以较小，是以损失图像质量为代价的。不过，因为 JPEG 的压缩算法比较合理，它对图形（图像）的损失影响并非很大。即便在 internet 上 VRML 三维（图形）图像技术日益成熟的今天，在表达二维图像方面，JPEG 仍有强大的生命力。

（8）TIG（tagged image file format）：文件体积庞大，但储存信息量大，细微层次的信息较多，有利于原稿色彩的复制。该格式有压缩和非压缩两种形式。Macintosh 机和 PC 机均支持该格式，在这两种硬件平台上移植 TIF 图形（图像）十分便捷。该格式最高支持的色彩可达 16M。

（9）EPS（encapsulated postscript）：用 postscript 语言描述的 ASCII 图形文件，在 Postscript 图形打印机上能打印出高品质的图形（图像），最高能表示 32 位图形（图像）。该格式分为 PhotoshopEPS 格式和标准 EPS 格式，其中标准 EPS 格式又可分为图形格式和图像格式。在 Photoshop 中只能打开图像格式的 EPS 文件。EPS 格式在 Macintosh 机和 PC 机上均常用。

（10）PSD（photoshop standard）：photoshop 中的标准文件格式，专门为 photoshop 而优化。

（11）CDR（coreldraw）：coredraw 的文件格式。另外，CDX 是所有 coreldraw 应用程序均能使用的图形（图像）文件，是发展成熟的 CDR 文件。

（12）IFE（image file format）：用于大型超级图形处理平台，比如 AMIGA 机，好莱坞特持大片多彩该图形格式处理。图形（图像）效果出色，包括色彩、纹理等可逼真再现原景。当然，该格式耗用的内存、外存等计算机资源也十分巨大。

（13）TGA（targe image format）：是 True vision 公司为其显卡开发的图形文件格式，创建时期较早，最高色彩数可达 32 位。VDV，PIX WIN ，BPX，ICB 等均属其旁系。

（14）PCD：Photo CD 格式，由 KODAK 公司开发，其他软件系统对其只能读取。

（15）MPT（Macintosh paintbrush 格式）或 MAC:Macintosh 机所使用的灰度图形（图像）模式，在 Macintosh paintbrush 中使用，其分辨率只能是 720×567。另外，Macintosh 机专用的图形（图像）格式还有 PNT，PICT，PICT2 等。

6.3 声音（音频）

声音是人们用来传递信息最方便和最熟悉的方式。声音携带的信息量大、精细、准确，以一个英语字符为例，表 6.1 列出其表达方式、数据量和信息量。

表 6.1 一个英语字符的表示

表示方式	数据量	信息
ASCII	1 字节	字符名称
点阵	7 字节	字符名称、字形字体
声音	2000 字节	字符名称、音高、音强、音长

人们期望计算机具有音频处理功能。把声、文、图集成一个系统中，并能交互地进行管理，这是对多媒体技术最基本的理解，最初的 MPC（多媒体 PC）是增加了音频处理的普通计算机。音频技术在多媒体中的应用极为广泛，例如：视频图像的配凌晨、配乐，没有声音的视频图像是不可接受的；静态图像的解说、背景音乐，可使图像充满生气，全景立体声可增强动感；语音电子邮件，听声如见人，倍增亲切感；可视电话、电视会议、协同工作（CSCW）等多媒体在能信中的应用都离不开音频；游戏中的音响效果，更加强了游戏的趣味性。
多媒体涉及到多方面的音频处理技术，例如：

- 音频采集：把模拟音频信号转换成数字序列，并存放于存储装置中。

- 语间编码/解码：把语音数据进行压缩编码、解压缩。

- 文--语转换：经过语言学处理，语音学处理，把计算机内的文本转换成声音输出。

- 音乐合成：利用音乐合成芯片，把乐谱转换成乐曲输出。

- 语音识别与理解：利用计算机辨识人说的话。

- 音频数据传输、音频视频同步、音频效果与编辑等。

6.3.1 基本概念

1.数字音频

模拟磁性录音技术的原理广泛地用于采集、播放各种各样的声音，如音乐、配音及特殊的声音效果。这种模拟录音方式是直接记录音频信号的波形，重放时用唱针扫描唱片槽纹或者用放音磁头来拾取信号。模拟磁性录音性能受电磁性能的影响较大。磁带的频率特性微小的变化，就会对音质产生影响。目前模拟录音动态范围可达 80bB（分贝，信噪比的单位）。若想进一步提高录音、放音的间质，只能求助于数字音频技术。

模拟音频技术中以模拟电压的幅度表示声音强弱。在数字音频技术中，把表示声音强弱的模拟电压用数字表示，如 0.5V 电压用数字 20 表示，2V 电压用 80 表示。

模拟电压幅度，即使在某电平范围内，仍然是连续和无穷多的，而用数字来表示音频幅度时，只能把无穷个电压幅度。用有限个数字表示，即把某一幅度范围内的电压用一个数字表示，这称之为量化。计算机内的基本灵敏制是二进制，为此我们也要把声音数据写成计算机的数据格式，这称为编码。

摸拟声音在时间上是连续的，而以数字表示的声音是一个数据序列，在时间上只能是断续的。因此当把模拟声音变成数字声音时，需要每隔一个时间间隔在模拟声音波形上取一个幅度值，称为采樾或抽样。该时间间隔为抽样周期（其倒数为抽样频率）。由此看出，数字声音是一个数据序列，它是由模拟声音经抽样、量化和编码后得到的。计算机、数字 CD、数字磁带（DAT）中存储的声单音。模/数转换器可以把模拟声音因此必须把模拟音频信号转换成有限个数字表示的离散序列，即实现单频数字化，在这一处于技术中，涉及到音频的抽样、量化和编码。

2.音频编码与标准

在音频数据的存储和传输中，数据压缩是必须的。通常压缩数据会造成音频质量的下降，计算量的增加。因此，人们在实施数据压缩地，在语音质量、数据率、计算量三方面要进行综合考虑。

为了减少数据率，专家们到力于压缩编码的研究，国际电报电话咨询委员会（CCITT，现在是 ITU-T）先后提出了一系列语音压缩编码的建议。从压缩方法讲，可分为波形编码、参数编码和混合编码。基于波形的压缩编码可获得高质量语音，但数据率不易降低。具体的方法有：μ（A）律压缩、自适应量化（APCM）、差值量化（DPCM）、自适应差值量化（ADPCM）、子带--自适应差值量化（SB-ADPCM）编码等,CCITT 基于上述方法提出的建立有 G.711（μ 律、A 律）、G.722（SB-ADPCM）。参数编码的参数有线性预测编码 LPC、共振峰等，这种编码方法的数据率低，但质量不易提高。近几年出现的混合编码，氢波形编码的高质量和参数编码的低数据率结合在一起，取得了较好的效果。比较成功的混合编码方案有：多脉冲线性预测编码（MPLPC）、码激励线性预测编码（CELPC）、规则脉冲激励 LPS 编码（RPE-LTP）等。这些算法和标准广泛用于公共电话网、ISDN、移动通信及多媒体技术中。

上述编码方法均有损声音的质量。科研人员正致力于此领域的研究，以期得到低码率、高质量的音频编码。

6.3.2 语音合成与语言合成

一般来讲，实现计算机语音输出有两种方法：一是录音/重放，二是文语转换。若采用第一种方法，首先要把模拟语音信号转换成数字序列，编码后，暂存于存储设备中（录音），需要时，再经解码，重建声音信号（重放）。录音/重放可获得高音质声音，并能保持特定人或乐器的音色。但所需的存储容量随发音时间线性增长。

第二种方法是基于声音合成技术的一种声音产生技术，它可用于语音合成和音乐合成。文语转换是语音合成技术的延伸，它能把计算机内的文本转换成连续自然的语声流。若采用这种方法输出语音，应预先建立语音参数数据库、发音规则库等。需要输出语音时，系统按需求先合成出语音基元，再按语音学规则或语言学规则，连接成自然的语声流。文语转换的参数库不随发音时间增长而加大，便规则库却随语音质量的要求而增大。目前世界上研制出汉、英、日、法、德等语种的文语转换系统，并在许多领域得到了广泛应用。

1.语音合成技术简介

语音合成已有多年历史，从合成采用的技术讲可分为发音参数合成、声道模型参数合成和波形编辑合成，从合成策略上讲可分为频谱逼近和波形逼近。

（1）发音器官参数语音合成：这种方法对人的发音过程进行直接模拟。它定义了唇、舌、声带的相关参数，如唇开口度、舌高度、舌位置与声带张力等。由这些发音参数估计声道截面积函数，进而计算声波。但由于人发音生理过程复杂性，理论计算与物理模拟之间的差异，合成语音的质量暂时还不理想。

（2）声道模型参数语音合成：这种方法基于声道截面积函数或声道谐振特性合成语音，如共振峰合成器、LPC 合成器。因内外也有不少采用这种技术的语音合成系统。这类合成器的比特率低，音质适中。为改善音质，发展了混合编码技术，主要手段是改善激励，如码本激励、多脉冲激励、长时预测规则码激励等，这样，比特率有所增大，同时音质得到提高。作为压缩编码算法，参数合成广泛用于通信、系统和多媒体应用系统中。

（3）波形编辑语音合成技术：20 世纪 80 世纪末 E.Moulines 和 F.Charpentier 提出基于时域波形修改的语音合成技术，在 PSOLA(pitch synchronous overlap add)方法的推动下，此技术得到了很大发展与广泛的应用。

波形编辑语音合成技术是直接把语凌晨波表数据库中的波形级联起来，输出连续语流。这种语音合成技术用原始语音波形替代参数，而且这些语音波形取自自然语音的词或句子，它隐含了声调、重音、发音速度的影响，合成的语音清晰自然。其质量普遍高于参数合成。

PSOLA 就是基音同步叠加，它把基音周期的完整性作为保证波形及频谱平滑连续的基本前提。该算法按以下三步实施：对原始波形进行分析，产生非参数的中间表示；对中间表示进行修改；将修改过的中间表示重新合成为语音信号，由于修改的侧面不同，又分为 TD-PSOLA，FD-PSOLA 和 LP-PSOLA。

这种语音波斯湾形编辑技术多用于文语转换系统中，现已有日、德、法、汉语的系统问世，采用这种技术应解决好以下问题：语音基元的选取、波形拼接过程中的平滑滤波、韵律修改以及语音学的分析和处理。

2.语音合成

让计算机说话，决不能是断续的音节，而应是连续自然，符合语言学规则的声音。根据人类的语言习惯，人在讲话前，首先要有意向（intention），然后在头脑中形成概念（concept），最后形成语言。目前，对人类大脑的高级神经活动了解甚少，语言合成仅处于文字到语音转换的层次上。文语转换（TTS）是一种智能型的语音合成，涉及到语言学、语音学、语音信号处理、心理学等多个领域。它按照多学科的需求，将文字转换成声音， 解决计算机语音输出的一种好方法。

目前世界上已研究出多种语言的 TTS 系统，如英语、法语、日语、汉语等。许多音频卡上都带有英语 TTS 软件，法国 CNET 实现的多语种 TTS 已在电话网中用于公共语音服务。

6.3.3 音乐合成和 MIDI

1.音乐合成

随着计算机的诞生，就实现了计算机奏乐；不过那仅仅是一个个单音，听起来十分单调。如何让计算机输出优美的音乐呢？最简单的方法是采用录音/重放方式。音乐的频带宽，需要提高其采样率和量化位数，因而数据量急剧增大。如用 44.1 kHz 的频率抽样，以每个样值 16 位表示某立体声音乐，那么每秒钟的数据为 176.4KB。由此可见，若以此种方式存储音乐，即使计算机的数据传输率允许，也只能存储很短时间的乐曲。

另一种方法是采取合成方式产生音乐。自 1976 年后就用调频（FM）音乐合成技术合成的音乐已经很逼真。1984 年又开发出一种更真实的音乐合成技术--波形表（wavetable）合成。

2.MIDI

MIDI 是音乐与计算机结合的产物。MIDI（musical instrument digital interface）是乐器数字接口的缩写，泛指数字音乐的国际标准，它始创于 1982 年。多媒体 Windows 支持在多媒体节目中使用 MIDI 文件。标准的多媒体 PC 平台能够通过内部合成器或连到计算机 MIDI 端口的外部合成器播入 MIDI 文件。利用 MIDI 文件演奏音乐，所需的存储量最少，如演奏两分钟乐曲的 MIDI 文件只需不到 8KB 的存储空间。

MIDI 标准规定了不同厂家的电子乐器与计算机连接的电缆和硬件。它还指定从一个装置传送数据到另一个装置的通信协议。这样，任何电子乐器，只要有处理 MIDI 信息处理器和适当的硬件接口都能变成 MIDI 装置。MIDI 间靠这个接口传递消息（message）而进行彼此通信。实际上消息是乐谱(score）的数字描述。乐谱由音符序列、定时和称作合成音色(patches)的乐器定义所组成，当一组 MIDI 消息通过音乐合成芯片演奏时，合成器解释这些符号，并产生音乐。

下面介绍一些有关 MIDI 的术语。

（1）MIDI 文件：存放 MIDI 信息的标准文件格式。MIDI 文件中包含音符、定时和多达 16 个能道的演奏定应义。文件包括每个能道的演奏音符信息：键、通道号、音长、音量和力度（击键时，键达以最低位置的速度）。

（2）通道（channels）：MIDI 可为 16 个通道提供数据。每个通道访问了一个独立的逻辑合成器。Microsoft 公司使用通道 1～1 0 作为扩展合成器，通道访问一个独立的逻辑合成器。Microsoft 公司使用通道 1 ～ 10 作为扩展合成器，通道 13～16 用作基本合成器。

（3）音序器（sequencer）：是为了 MIDI 作曲而设计的计算机程序或电子装置。音序器能够用来记录、播放、编辑 MIDI 事件。大多数音序器能输入、输出 MIDI 文件。

（4）合成器（synthesizer）：利用数字信号处理器或其他芯片来产生音乐或声音的电子装置。数字信号处理器产生并修改波形，然后通过声音产生器和扬声器发出声音。合成器发声的质量和声部取决于以下因素：合成器能够同时播入的独立波形的个数、控制软件的能力、合成器电路中存储空间。

（5）乐器（instrument）：能产生特定声音的合成器。不同的合成器，乐器音色号不同，声音质量不同。不同乐器使用的音色号不同，输出的声音是有差异的。

（6）复音（polyphony）：合成器同时支持的最多音符数。如一个能以六个复音合成四种乐器声音的合成器，可同时演奏分布于四种乐器的六个音符，它可能是四个音符的钢琴和弦、一个长笛和一个小提琴的音。

（7）音色（timbre）：音色指的是声音的音质。音色取决于声音频率的组成。在非正式用法中，综拽的是与特定乐器相关的特定声音，如低音提琴、钢琴、小提琴的声音均有各自的音色。

（8）音轨（track）：一种用通道把 MIDI 数据分隔成单独组、并行组的文件概念。0 号格式的 MIDI 文件把这些音轨合并成一个，1 号格式的 MIDI 文件维持不同的音轨。

（9）合成音乐映射器（patch appear）：是一种软件。为了适应 Microsoft MIDI 的合成音色，分配表规定了合成音色的编号。合成音色映射器更为特定的合成器重新分配乐器合成音色的编号。多媒体 Windows 的映射可将乐器的合成音色映射到任意 MIDI 装置上。

（10）通道映射（channel mapping）：通道映射把发送装置的 MIDI 通道号变换成适当的接收装置的通道号。例如编排在 16 号通道的鼓乐，对于仅接收 6 号通道的鼓来说，就被映射成 6 号通道。

3.MIDI 和多媒体 PC

MIDI 规范允许 MIDI 装置以预选说明的方式通信。为了提供单电缆连接和通信端口标准，关键之一是物理连接的标准化。MIDI 标准中规定 MPC 包括一个内部合成器和标准 MIDI 端口。
MIDI 装置应有一个或多个下列端口，MIDI In,MIDI Out 和 MIDI Thru。每种端口有特定的用途，如发送、接收或在 MIDI 装置间转发 MIDI 消息。这种设计允许同时控制所连接的多个 MIDI 装置。各端口的功能简述如下：

（1）MIDI In（输入口）：接收到其他 MIDI 装置传来的消息.

（2）MIDI Out（输出口）：发送某装置生成的原始 MIDI 消息。向其他设备发送 MIDI 消息。

（3）MIDI Thru（转发口）：传送从输入口接收的消息到其他 MIDI 装置。向其他设备发送 MIDI 消息。

上述 MIDI 端口都支持标准的 MIDI 电缆连结。MIDI 电缆由屏蔽的双绞线及连接电缆两端的五针 DIN 插头组成。MIDI 乐器间的连接如图 6.3 所示。

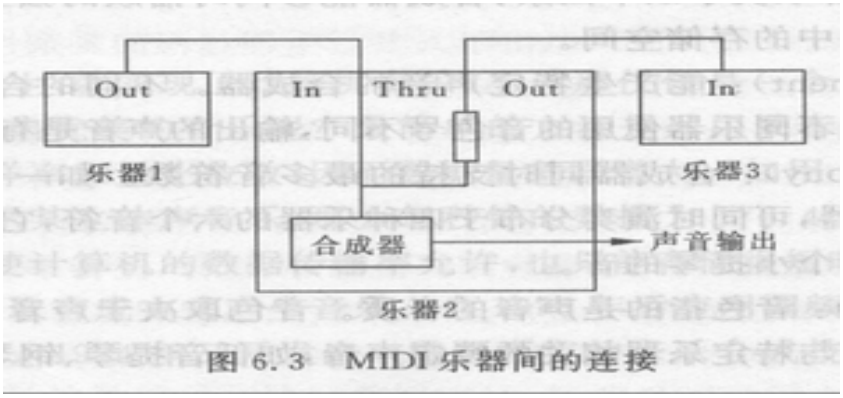


图 6.3 MIDI 乐器间的连接

除了物理连接外，MIDI 规范中还规定了用于 MIDI 装置彼此通信用的标准消息。这些消息指定一个或多个 MIDI 装置来定义各产生音乐的事件。消息的内容还定义了弹奏的音符、乐器 的变换等事件。定义和产生歌曲的 MIDI 消息和数据存于 MIDI 文件中。使用音序器建立 MIDI 文件，它获取 MIDI 消息，并把它们存于文件中。演奏 MIDI 文件时，音序器把 MIDI 消息从文件送到合成器，合成器把这些消息转换成特定乐器、特定音高和时长的声音。合成器用 DSP 或其他芯片产生并修改波形，得以合成音乐和声音，并通过声音发生器和扬声器送出去。

4．什么时候使用 MIDI

有了波形音频的众多功能，那为什么还要使用 MIDI？这是因为：MIDI 文件是一系列指令，而不是波形，它需要的磁盘空间非常少；并且预先装载 MIDI 文件比波形文件容易得多。这样，设计多媒体节目时，可以指定什么时候播放音乐，将有很大的灵活性。

以下几种情况，使用 MIDI 谱曲比使用波形音频更合适：

·需要播放长时间高质量音乐。比如想在硬盘上存储的音乐在大于 1 分钟，而硬盘又没有足够的存储容量。

·需要以音乐作背景音响效果 ，同时从 CD-ROM 中装载其他数据，如图像、文字的显示。

·需要以音乐作背景音响效果，同时播放波形音频或实现文语转换，以实现音乐和语音同进输出。

6．3．4 音频文件格式的比较

在此，对七种最为流行的声音文件效果作一简要比较：

1．WAVE，扩展名 WAV

该格式记录了声音的波形，故只要采样率高、采样字节长、机器速度快，利用该格式记录的声音文件能够和原声基本一致。从理论上说，采样率达 44kHz（每秒采样 44000 次）、采样字节长度 达 16 位的音质已能和常规 CD 唱片相当。WAVE 可以不对数据进行压缩，这样只要拾音过程"把好关"，制作出来的声音质量可以达到专业级的标准。

WAVE 的惟一缺点就是文件太大，毕竟它要把声音的每个细节都记录下来，而且不压缩。如果采样率为 44kHz，那么每一秒钟就有 44K×16×2（立体声）=1441792 位产生，那么，一张 650KB 的空白光盘最多也只能容纳五六十分钟的节目。

2．MOD，扩展名 MOD，ST3，XT，S3M，FAR，669 等。

该格式及其播放器大约起源于 20 世纪 80 年代初，原先是作为"软声卡"问世的--利用 Modplayer 可以通过机器自带喇叭或通过 LPT 口自制"声卡"直接播放乐曲。MOD 只是这类音乐文件的总称，因为最初的文件扩展名是 MOD，后来逐渐发展产生了 ST3，XT，S3M，FAR，669 等扩展格式，而其基本原理还是一样。

该格式的文件里不仅存放了乐谱（最初只能支持 4 个声道，到现在已有 16 甚至 32 个声道的文件及播放器了）而且存放了乐曲使用的各种音色样本。由于制作人创作歌曲使用的音色样本同听众回放文件时使用的音乐样本完全相同，故这样的文件有几个显著优点：

（１） 回放效果明确：相对于 MIDI 来说，MOD 对回放设备的要求较低，只要 CPU 处理速度够快，不丢失信息，编曲者是能够预测回放效果的（而 MIDI 则完全依赖于回放的 MIDI 音源）。

（２） 音色种类永无止境：由于 MOD 的音色样本制作非常简单，CD 唱片、WAVE 音乐、MIDI 音乐、MIC 话筒输入、LINE IN 线路输入等所有方式产生的声音都能采样为音色样本（甚至人声），故采用 MOD 音乐可以很轻松地引入民族乐器、专业效果等音色。此外，MOD 文件的制作不需要乐器键盘，可以完全利用计算机键

盘一字一句输入，它有点像"八音盒"，只要在空白纸片上打小孔就可以了（如果这么比喻的话，那么孔的大小、开头也是有讲究的）。

3．MPEG-3，扩展名 MP3

虽然 MPEG-3 压缩较大（一张 MP3 CD 可容纳 10 张 CD 唱片歌曲），但其实际音质并不是非常满意。可能有人要说，MP3 能达到 44kHz 和 16 位，为何不能和 CD 间质媲美？其实单从 44 kHz 这类指标来看并不能说明实际效果的好坏。MP3 的音质也一样，它的确可以发出高得耳朵都听不见的声音，但"该发的声音并没有全部发出来"。

在网络、可视电话通信方面，MP3 大有用武之地，如果要它和 CD 唱片相比，则有些令人遗憾。由于本质不同，所以它没法和 MOD、MIDI 相提并论，只能这么说，从 HIFI 角度讲，MP3 有损失，而 MOD、MIDI 则没有。

4．Real Audio, 扩展名 RA

和 MP3 相同,它也是为了解决网络传输带宽资源而设计的,因此主要目标是压缩比和容错性,其次才是音质.

Real Audio 压缩比很大,一首《狮子王》主题曲"Can you feel the love tonight"只有约几百 K。3.0 和 4.0 的文件支持 28.8K、56K 甚至更快的网络传输，因而其音质也比以前高得多。4.0 的文件标榜"near CD quality"（接近 CD 音质），不过从另一角度来说，这永远也只是"near (接近)，因为它是压缩的，有损失。拿它和 MP3 相比，从 HIFI 角度来说，可能 MP3 更好。

5．Creative Musical Format,扩展名 CMF

这是 Creative 公司的专用音乐格式。它和 MIDI 差不多，只是音色、效果上有些特色，专用于 FM 声卡。不过其兼容性差，"不成气候"，且效果感觉像音乐门铃，无法和别的格式相提并论。声霸卡使用音频文件格式为 VOC 文件。

6．CD Audio 音乐 CD ，扩展名 CDA

CDA 格式就是唱片采用的格式，又叫"红皮书"（Red Book）格式，记录的波形流、绝对的纯正、HIFI。音乐 CD 的回放采样字节都是 16 位，现在有些厂家在录制 CD 时采用 20 位录音，这样就产生了一些耳朵听不到但大脑感觉得到的波形，可谓 CD 中的精品。YAMAHA 前一时期推出的 Pro Bit 技术也正是把常规的 16 位唱片通过计算"变"出了丢失的每个比特。
CDA 的缺点是：无法编辑，文件太大。

7．MIDI，扩展名 MID

随便哪个专业录音棚里都能看到 MIDI 设备的影子。作为音乐工业的数据通信标准，MIDI 可谓是一种非常专业的语言，它能指挥各音乐设备的运转，而且具有

统一的标准格式,能够模仿原始乐器的各种演奏技巧甚至无法演奏的效果。因此,MIDI 实际上是种产业标准，它的科学性、兼容性、复杂程序等各方面当然远远超过本文前面介绍的所有标准（除交响乐 CD、Unplug CD 外，其他 CD 往往都是得用 MIDI 制作出来的）。

MIDI 作为世界通用标准已很成熟，所谓 General MIDI 就是最常见的通行标准，此外 Roland 公司的 GS（general standard）、YAMAHA 公司的 XG 等标准也成为购买音源时要考虑的项目之一。由于标准化，MIDI 文件的长度非常小，一首歌一般就几十 K，复杂的稍多些，但远比 MOD 要小。

为了发挥 MIDI 短小的长处，避免对回放设备依赖的短处，现在在网络上开始出现了"软波表"之类的"软音源"。采用专业音源的波表，利用 CPU 对网络上传来的短短的 MIDI 数据进行回收，其效果能够被制作者预测，因而在音源硬件设备未向大众普及的时代，"软波表"必将处于较有利的地位。

MIDI 的另一个"缺点"就是不能记录人声等声音。它保能将人声作为一种乐器储存在音色库中，要么就是"啊"音，要么就是"哇"音……总之不能令乐器唱歌、讲话。这并不是 MIDI 的错，因而 MIDI 只是个音乐标准，不是录音标准，因而无法取代 Real Audio.

总之,如果有专业的音源设备,那么要听同一首曲子的 HIFI 的程度依次是:

原声乐器演奏>MIDI>CD 唱片>MOD>所谓声卡上的 MIDI>CMF,MP3 及 PA 要看它的节目源是采用 MIDI,CD 还是 MOD 了.

6.4 视频(动画)

动态图像,包括动画和视频信息,是连续渐变的静态图像或图形序列沿时间轴顺次更换显示,从而构成运动视感的媒体.当序列中每帧图像是由人工或计算机产生的图像时,我们常称为动画;当序列中每帧图像是通过实时摄取自然景像或活动对象时,我们常称为影像视频,或简称视频.

动态图像演示常常与声音媒体配合进行,二者的共同基础是时间连续性. 一般意义上谈到视频时,往往也包含声音媒体.但在本节中,视频(动画)特指不包含声音媒体的动态图像.

6.4.1 基本概念

在视频摄像机获取录像带后，通过一定的硬件转换设备，可以把模拟信息转变成数字信息存房再计算机中，经过软件的播放，又可以重新看到录制的信息。对于这一过程，我们都比较熟悉，这里不详细讨论。下面主要介绍动画的基本概念。

由迪斯尼电影公司和好莱坞生产的"美女与野兽"、"灰姑娘"及其他许多动画故事片都是色彩鲜艳、声音丰富的动画，这是在没有可遵守的自然界限的媒体空间中

进行交流的能力。就是说，在电影中，房子看起来就像房子，人看起来就像人，而在动画中，可以基于制造者、交流者的视觉和理解来创造物体，可以扭曲、翻转、旋转物体，涂上炫目的色彩等，以一种对真实物体来说并不常见的方式来表现物体。

动画在娱乐业存在已经超过 50 年的历史了，最终，随着超级个人计算机和大容量数据存储器的出现，动画进入商业领域。某些情况下，也可以把动画功能加到现有的应用软件上。

什么是动画?动画就是能过以 15～20 帧每秒的速度(相当接近于全运动视频帧速)顺序地播放
静止图像帧以产生运动错觉。因为能足够长时间地保留图像以允许大脑以连续的序列把帧连
接起来，所以能够产生运动错觉。实际上，迪斯尼就是通过绘制大量给人以运动感觉的递增图片来实现卡通制作的。动画影片含有一系列的帧，每帧中都含有运动递增的物体。有时不必使物体运动，而是一帧一帧的颜色和背景，就会有一种物体运动的感觉。实际上，有许多方法来获得不同层次的动画。

可以通过在显示时改变图像生成简单的动画。最简单的方法是在两个不同的帧之间反复。这种方法对于指示"是"或"不是"的情况来说是很好的解决方法，例如，在两个图符间作反复以指示通信活动正在进行之中。另一个例子是两幅有关胶片的位图，一幅显示了胶片卷起来，另一幅显示胶片打开，可以通过这两幅位图之间的反复来生成一种动画化的状态以指是否出现了未读的视频文件。可以使用计时器来控制两幅图像之间特定的间隔反复，以便确保用户能看到变化。计时器以 ms 增量来度量间隔，这样，在上述例子中可以使用设定为 500ms 的间隔以使胶卷 0.5s 打开或关闭。

采用循环的形式播放几个图像帧以生成旋转的效果，这是另一种动画制作的方法。例如，旋转一个地球仪以指示正进行的动作，动画中含有几个以循环形式显示的帧。由于动画中由单个帧组成，因此呆在任何时刻暂停或恢复回放，应用软件能递增帧的号码以跟踪动画中的当前帧。注意，如果可以依靠计算时间来获得较好的回放，也可以用计时器来控制动画。
当然，构成动画的方法还很多，就不一一介绍了。

6.4.2 视频格式

视频信息在计算机中存放具体格式有很多，这里介绍比较流行的两种。

1. 苹果的公司的 Quicktime

苹果的公司的 Quicktime 产品采用了面向最终用户桌面系统的低成本、全运动视频的方式，在软件压缩和解压缩中也开始采用这种方式。向量量化是 Quicktime 的软件压缩技术之一，它在最高为 30 帧/s 下提供的视频分辨率是 320×240，而且不用硬件帮助。向量量化预计可成为全运动视频的主要技术，向量量化方法达到的压缩比例为 25：1～200：1。其视频信息采用 MOV 文件格式。

2. 微软的 AVI

类似于 Quicktime，微软公司的音频视频交错（AVI）也是桌面系统的上的低成本`低分辨率的视频格式，Quicktime 是操做系统的一部分，而 AVI 则设计为与系统分离的分层模块。作为只用软件的方法，AVI 可在 160×120 的视窗中以 15 帧/s 回收视频并可带有 8 位的声音。也可以在 VGA 或超级 VGA 监视器上回收。与超过 320 线的 VCR 分辨率相比，这一分辨率明显低于正常电视信号的分辨率。

AVI 很重要的一个特点是可伸缩性，使用 AVI 算法的性能依赖于它一起使用的基础硬件。AVI 包括了几种基于软件的压缩和解压缩算法，其中某些算法被优化用于运动视频其他算法则被优化用于静止视频。用 AVI 达到的质量与 Quicktime 相似，尽管它们是基于不同的技术；在外表上，它们看起来也很相似。

6.5 多媒体数据压缩和编码技术标准

近年来，与数据压缩有关的技术研究支持了多媒体的发展，使原来不能实现的应用变为现实。ISO、IEC、ITU 等国际已制定和正在制定各种通用压缩编码标准。

6.5.1 H.261

1990 年 CCITTA(ITU-T)通过了国际视频会议标准，包括用于音频服务的 P×64Kbps 的 H.261 视频编码解码器(也称 P×64 标准)。这一标准预期的应用是可视电话和视频会议系统。因此，含有此标准的系统必须能实时地按标准编码和解码。可视电话应用的 P 值范围是 1～2(基于最高为 128Kbps 的 ISDN)。用多线可使 P 值增加。

CCITT 采用 CIF(公共中间格式)和 QCIF(四分之一 CIF)作为可视电话的视频格式，所有解码器都要求在 QCIF 水平上操作，在 CIF 水平的操作是任选的。在大约 30 帧/s 的情况下，(能支持的最大值是 29.97)，未压缩的 CIF 包含 36.45Mbdps 的信息，而 QCIF 包含 9.115Mbps 通道要求。即使传送 QCIF 信号也必须进行显著压缩，10 帧/s 时，QCIF 对 1.5Mbps 的信息。即使传送 QCIF 信号也必须进行显著压缩，10 帧/s 时，QCIF 对 1.5Mbps 通道要求 3：1 的压缩率，但对 P×2 通道(128Kbps)就要求高达 24：1 的压缩率。通常 CIF 被推荐用于超过 256Kbps 的传送。

CIF 和 QCIF 采用分层块结构来编码数据，它们包括图像群块(GOB)和宏块(MB)。每个 MB 由四个 8×8 亮度块和两个色度块组成(普通 NTSC 信号由一个亮度信号和两个色度信号组成)。对于 QCIF 一个 GOB 由 3×11 个 MB 组成，对于 CIF，则是 12×11 个 MB 组成。与 JPEG 类似，H.261 使用两种类型的压缩：一帧中的有损压缩(基于离散余弦变换 DCT)和用于帧间压缩的无损熵编码。

H.261 的编码器采用带有运动估计的 DCT 和 DPCM(差分脉冲编码调制)的混合方式。DPCM 在帧内压缩过程中是不运行的(使用 DCT)，但在帧间压缩过程中处于运行状态，并使用了基于运动估计的预测概念。它将当前帧中每个 MB(只是亮度成分)与下一帧中邻近的 MB 作比较。如果差别低于预选定义的阈值，就不传送差分数据。如差别高于阈值，那么差分 DCT 就被传送、线性量化，然后与运动矢量信息一起送到视频复合编码器中。线性量化器使用步进算法，它可以按图像质量和编码效率调节。

H.261 也定义了数据格式。每个 MB 都包含块的 DCT 系数(TCOEFF)，TCOEFF 之后是 EOB(固定长度块结束标记)。每个 MB 都由数据和 MB 标题组成，GOB 由包装了 MB 序列数组的 GOB 标题组成。图像层由包装 GOB 序列数组的图片标题组成。

H.261 与 JPEG 及 MPEG 标准间有明显的相似性，要注意的关键区别是 H.261 是为动态使用而设计的，并提供完全包含的组织和高水平的交互控制。

6.5.2 JPEG

自 1982 年 6 月以来，ISO 的 WG8(Working Group 8)就致力于静止图像压缩和解压缩算法的标准化工作。到 1987 年 6 月，先后提出了十种算法，分别针对彩色和灰度静止图像。经过反复比较，一种基于 DCT 的算法脱颖而出，被广泛采纳。这就是 JPEG(joint Photo graphic Coding Experts Group)标准的由来。1992 年 JPEG 成为 ISO 国际标准。

6.5.3 MPEG

MPEG 是 Moving Pictures Experts Group(动态图像专家组)的英文缩写，这个专家组始建于 1988 年，专门负责为 CD 建立视频和音频标准，其成员均为视频、音频及系统领域的技术专家。由于 ISO/11172 压缩编码标准是由此小组提出并制定，MPEG 由此扬名世界。对于今天我们所泛指的 MPEG-X 版本，是指一组由 ITU(International Tele communications Union)和 ISO 制定发布的视频、音频、数据的压缩标准。

MPEG 的缔造者们原先打算开发四个版本：MPEG-1～MPEG-4，以适用于不同带宽和数字质量的要求。后由于 MPEG3 被放弃(其实并未完全放弃，现在听到的 MP3 的曲子，就是按此标准压缩的)所以现在只有三个版本的 MPEG：MPEG-1，MPEG-2，MPEG-4。

总体来说，MPEG 在三方面优于其他压缩/解压缩方案。首先，由于开始它就是作为一个国际化的标准研究制定的，所以具有很好的兼容性；其次，MPEG 能够比其他算法提供更好的压缩比，最高可达 22：1。更重要的是，MPEG 在提供高压缩比的同时，对数据的损失很小。

1.MPEG-1

MPEG-1 制定于 1992 年，为工业级标准而设计，可适用于不同带宽的设备，如 CD-ROM，CD，CD-I。它可针对 SIF 标准工业帧分辨率(对于 NTSC 制为 352×240，对于 PAL 制为 352×288)的图像进行压缩，传输速率为 1.5Mbps，每秒播放 30 帧，具有 CD(指激光唱盘)音质，质量级别基本与 VHS 相当。MPEG 的编码速率最高可达 4Mbps～5Mbps，但随着速率的提高，其解码后的图像质量有所降低。

MPEG-1 也用于数字电视网络上的视频传输，如非对称数字用户线路(ADSL)、视频点播(VOD)以及教育网络等。同时，MPEG-1 也可用作记录媒体或是在 Internet 上传输音频。

2.MPEG-2

MPEG-2 制定于 1994 年,设计目标是高级工业标准的图像质量以及更高的传输率。MPEG-2 所能提供的传输率在 3Mpbs～10Mbps 之间，其在 NTSC 制式下的分辨率可达 720×480，MPEG-2 也可提供并能够提供广播级的视像和 CD 级的音质。MPEG-2 的音频编码可提供左右中及两个环绕声道,以及一个加重低音声道和多达七个伴音声道(DVD 可有八种语言配音的原因)。由于 MPEG-2 在设计时的巧妙处理。使得大多数 MPEG-2 解码器也可播放 MPEG-1 格式的数据，如 VCD。

同时，由于 MPEG-2 的出色性能表现，已能适用于 HDTV，使得原打算为 HDTV 设计的 MPEG-3，还没出世就被抛弃了(MPEG-3 要求传输率在 20Mbps-40Mbps 之间，但这将使画面有轻度扭曲，目前用于 MP3 音乐)。除了作为 DVD 的指定标准外，MPEG-2 还可用于广播、有线电视网、电缆网络以及卫星直播(direct broadcast satellite)提供广播级的数字视频。

MPEG-2 的另一特点是，可提供一个较广的范围改变压缩比，以适应不是画面质量、存储容量以及带宽的要求。

对于最终用户来说，由于现存电视机分辨率的限制，MPEG-2 所带来的高清晰度画面质量(如 DVD 画面)在电视上效果并不明显，而其音频特性(如加重低音、多伴音声道等更引人注目)。

3.MPEG-4

MPEG 专家组的专家们正在为 MPEG-4 的制定努力工作。MPEG-4 标准主要应用于视像电话(videophone)、视像电子邮件(video email)和电子新闻(electrinic news)等，其传输速率要求较低，在 4800bps-6400bps 之间，分辨率为 176×144。

MPEG-4 利用窄的带宽，通过帧重建技术，压缩和传输数据，以求以最少的数据获得最佳的图像质量。

与 MPEG-1 和 MPGE-2 相比，MPEG-4 更适于交到 AV 服务以及远程监控 。MPEG-4 是第一个使观众由被动为主动(不再只是观众，允许加入其中，即有交互性)的动态图像标准；它的另一个特点是综合性，从根源上说，MPEG-4 试图将自然物体与人造物体相融合(在视觉效果意义上)。

MPEG-4 的设计目标还有更广的适应性和可扩展性。

6.5.4 DVI

DVI 视频图像压缩算法的性能与 MPEG-1 相当，即图像质量可达到 VHS 的水平。压缩后的图像数据率约为 1.5Mbps.应用 Intel 公司生产的 i750 芯片组即 82750PB 和 82750DB 可实时完成 DVI 视频图像的编码和解码算法。

为了扩大 DVI 技术的应用，Intel 公司又推出了 DVI 算法软件解码算法，称为 Indeo 技术。它能将未压缩的数字视频文件压缩为 1/5～ 1 /10。Indeo 技术已被附加在某些产品中，如微软公司 Video for Windows 和苹果公司的 Quicktime.

Indeo 技术使用了多类"有损"和"无损"压缩技术。Indeo 技术在视频捕获卡记录同时就实时地对它进行压缩，因此未压缩的数据不需存在盘上。从视频摄像机、VCR 或激光盘上接收到的以任何标准格式(如 NTSC 存在的视频)都由视频捕获卡(如 Intel 的 Smart Video RecorderBoard)转换为数字格式。

这些数字信号经过下列类型的压缩：

• YUV 采样，将像素面积精简压缩到平均彩色值；

• 像素差分和时间压缩，通过只存储像素或帧之间的变化信息(向量量化)来压缩数据；

• 游程编码，压缩量化的向量信息；

• 可变内容编码，将可变数量的信息压缩固定数目的位。

上述这些种类的压缩是由软件压缩技术完成的。

压缩的数字化视频文件与音频信息组合成为标准的文件格式，如微软的 AVI(音频视频交错)或苹果公司的 Quicktime,并存于硬盘。组合的文件可被传输和分配用于回放的和编辑。用于回放时，文件必须分隔为音频和视频部分。视频通过若干方法进行解压缩(是压缩步骤的逆
过程)来获得未压缩数字视频原始数字素表示。

实际上，三种因素将影响视频的性能，它们是微处理器速度、回放视窗尺寸(用像素表示)以及帧率。较小的回放视窗将产生更平滑更自然的视频图像。更快的微处理器能支持较大的回放视窗和更高帧率。Indeo 技术是可伸缩的，即它随着处理能力的提高可以提供给用户更快
的帧率。

6.6 多媒体创作工具及发展方向

多媒体创作系统是多媒体电脑技术中的一大重要组成部分。它介于多媒体工作平台与应用软件之间，是支持应用开发人员进行多媒体应用软件创作的工具，故又称之为称多媒体创作工具。

6.6.1 多媒体创作工具概述

多媒体创作工具产生的初衷是：为不懂编程的应用人员制作应用软件提供一种便利的工具。这些应用人员都是某些领域中的专家，如教育家、文学家或导弹专家等等。这些人员的特点是：对自己的专业了如指掌，但对电脑编程不甚了解。如他们因工作需要，要制作如 CAI 类、模拟类、百科类以及电子书类等软件产品，这就必须请软件工程人员来编制，由于双方存在 沟通、理解和维护等问题，作出来的产品往往不尽如人意，为此需开发一种工具，借助这种工具，使就用人员不用编程也能做出很优秀的多媒体软件产品，这种工具称之为多媒体创作工具。用这种工具制作出来的软件产品有一个特别的名字-节目(title)

由此可知，多媒体创作工具必须担当起可视化编程的责任，它必须具有概念清楚、界面简洁、操作简单、功能伸缩性强等特点。初始的多媒体创作工具都有自己擅长 CAI，有擅长制作数据库查询系统等等。但从目前发展来看，节目的表现范畴越来越广泛，表现手法越来越灵活，最终用户的功能要求越来越复杂。相应地，各种版本的多媒体创作工具了为了适应潮流，功能也是越来越复杂，不再满足于只在某一领域里具有某种特长。从功能上看，它们似乎什么都做，因为大多数强大的多媒体创作工具或多或少都自带语言功能，甚到是外部程序接口。

6.2.2 多媒体创作工具的发展方向

多媒体创作工具是能够用来集成各种媒体，并可设计阅读信息内容方式的软件。它的发展历程不过七八年，但其功能却日趋完善与成熟。从当前发展的趋势来看，以下几个方面是一个优秀的多媒体创作工具不可缺少的功能，至少需努力的方向。

1.编辑能力及环境

多媒体编辑的主要功能是支持用户对节目所需的外部媒体资料的生成、增删、修改与管理能力，可能包括如下工具：动画编辑器、文书编辑器、影像编辑器、音频编辑器、视频编辑器、面向对象的媒体数据库管理系统(OODBMS)。初期的多媒体创作系统只能载入这些类型的数
据，并作简单的增、删、拷贝等编辑，如果涉及到修改，则必须使用相应其他软件修改。如今，集成化的多媒体创作工具已出现，它带有图形、动画、音频、视频、等资料的生成模块，结构小巧实用，方便用户制作。如果创作工具中加入目标连接和嵌入(OLE)功能，使用户在编辑媒体对象时能及时更新节目中的内容，这会大大提高制作效率。编辑环境主要是指媒体元素编入节目程序流的能力，它要求具有对基本的信息流控制操作能力。基本的信息流操作是指循环、条件分支

以及使用变量等价、布尔逻辑和简单数学计算进行媒体数据控制和管理等。这些功能不再是通过用户写程序实现，而是在系统中通过可视化的界面，帮助用户使用简便操作来达到相同的效果。

2.媒体数据输入能力

由于多媒体应用经常需要处理传统的和新的媒体，这就需要多媒体创作工具有处理静态和动态多媒体的能力。而且支持的格式越多越好。最基本的要求是可处理具有两种或两种以上格式的位图文件能力，这些格式可能包括"BMP，DIB，PCX，TIF，GIF，TGA，WMF，PICT，IMG
或 JPEG 等；FLIC，MMM，AVI，MPG，MOV 和 DAT 等动画文件；WAV，MID，CMF 和 VOC 等音频文件；数据库文件等，此外还有文本格式的文件。支持外部数据的格式越多越好，以免制作时因格式不对而需找其他软件转换造成麻烦。

3.交互能力

多媒体创作工具能提供多大程度的交互功能是评价该工具优秀与否的重要指标之一。最普通的交互方式是通过设定按钮来实现的，用户可以设定触发按钮来连接某一事件，达到交互目的。电子书中超文本是不可缺少的交互功能，多媒体创作工具中可以设定文字对象中的单字或短语含有活动的信息链，使用者通过触发这些单字或短语，使信息链发生作用而引发一个动作或跳到相关的数据单元。超文本以有下几种类型：动作类(激发一段内部程序或一组时基数据对象)、索引类(激发一内部索引功能)、解释类、调用类和跳转类等。

这几年，随着多媒体创作工日益成熟，交互方式的花样也增加了，例如不可见区域的交互、菜单交互、可视媒体对象(如圆、直线、图像)的交互、文字输入交互以及程序交互等等。引发交互的手段有：条件交互、接触交互、拾取交互、判结果交互、按键交互、主题词交互等。随着多媒体创体创作系统对硬件控制能力的加强，语音交互、触控交互、三维立体触控交互以及未来的智能判断交互等新技术会日益普及起来。

4.功能扩充能力

新一代多媒体工具非常注重功能的可扩充性。老一代的多媒体工具因为面向不懂编程的人员，采用提可视化编程，学习虽然简单，但用户马上就发现系统提供的这些功能远远不能应付日益复杂的节目设计效果，于是一些多媒体创作工具连入了描述语言。这种描述语言复杂度类似于 Basic 语言，与系统相辅相成。这种描述语言使系统的功能有了提升，基本满足了用户制作节目的需要。为了满足高层次节目的需要，一些多媒体创作工具提供了外部程序接口，可以连入 DLL，VBX 等高级对象或组件。这样，多媒体创作工具实现了三级用户开发环境，这三级用户是：不懂编程的普通用户(使用系统提供的工具制作节目)、稍懂程序的用户(结合描述语言制作节目)、富有编程经验的用户(通过外部程序接口直接调用程序)。通过这种方式，使得多媒体创作工具的功能可以不断地扩充。

有的多媒体创作工具还提供现场录制方式，如 ToolBook 系统可自动将用户的操作过程转换为描述语言。好的多媒体创作工具支持描述语言程序的内部和外部格式相互转换。它采取面向对象的编辑观念，用户是通过对系统中原有的类定义编辑对象，为了扩充系统中不支持的类，用户可以通过描述语言自定义需要的类以便加强系统功能；这些类将完整地支持面向对象的思想，支持类的继承、派生和重载等特性。未来的多媒体创作系统将只提供一个开发框架(framework)，使用户在使用的过程中，可以不断加强系统类别和函数或组件的功能，最后逐渐 将整个系统变为真正面向用户自己的专用系统。

5.调试能力

目前多媒体创作系统的调试功能还只停留在：用户可设定放映节目的范围、发现错误后可及时中断、单放、快放、报告程序的出错点和错误类型。未来的调试工具，应加入可以设定多断点、逆向回放、自动分支覆盖调试等功能，在调试过程中应显示相应的调试信息，中断时可察看环境状态的变化，有自动查错和定位错误的功能等。总之，调试环境在多媒体创作系统中变得越来越重要，特别是在语言功能被加入系统中后。

6.动态数据交换能力

动态数据交换功能是指：使用变量给某些媒体构件赋予属性，并且可以通过程序改变变量， 从而达到动态改变媒体属性的目的。此外，如果是基于数据库的多媒体查询软件，那么数据应可动态地通过读取数据库中的资料及时更新。在此基础上，如再加入远程网络传输功能，
在节目每次演示前都重载数据，随时都可以显示最新信息，这对于制作企业及金融业等多媒体软件具有非常巨大的实用价值。

7.数据库功能

使用多媒体创作工具制作百科类、查询类等节目时，数据库作为辅助软件是必不可少的，节目中的数据可以通过数据库文件取得。除了传统的五种数据库字段外，还应加上图像、声音、动画等一般数据库中无法处理的媒体资料。

好的多媒体创作工具应具有数据库的各式查询功能：查找、排序、更新、删除、新增记录等，并可根据使用者的需要抽取所需的记录。

8.网络组件及模板套用能力

大型多媒体节目是多种专业人才通力合作的产品，如何使用电脑网络来支持一组人进行协同创作，是未来多媒体合作工具在发展中必须考虑的问题。这种能有效支持一组人通力制作一个节目的创作工具称为组件版本(group ware version)。该环境支持异构电脑系统之间的互联和资料传递，使用各种类型的电脑，以及在不同操作系统下工作的小组人员，均可在电脑网络的基础上，通过该组件版本建构一个多媒体创作的工程开发环境，这个环境不仅具有个人版本所应有的全部支

持性能，还应具有如下性能：任务分割与进度控制，成员通信，并发编辑控制与冲突的解决，产品编辑权限控制以及版本更新、备份等控制。

模板功能是提高用户编辑效率的一个重要手段，方便用户大量制作某一类型的节目。通过系统提供的或用户制作的一个产品模型或框架(product frame)，用户只要将需要变化的数据套入这个模板或框架中，修改或填充其中的内容，即可方便地制作同一类型的另一新节目。
目前大多数创作工具只提供简单的已制作好的模板，供用户在上面修改，但如何大规模地批量更换模板内的数据资料，并能做到速度快而操作简单，目前还是各个多媒体创作工具应重点考虑的问题。

9.其他

以上是多媒体创作工具着重发展的部分，但是还有许多其他功能也是目前或未来不可缺少的，例如电子书中的笔记(note)、书签(bookMark)、回溯(back)、历史(history)等功能，以及在线测试功能(节目与题库系统相连，可通过一条信息链触发在线测试功能，并做出相应的回馈响应)、产品编译功能(节目制作完成后，创作工具可以自动根据用户有用的资料，以用户要求的方式自动生成一个可以独立放映的子节目)、预加载功能(在节目运行过程中，为了支持慢速的外部存储设备，可以预先载入较大的数据)、多国语言版本的制作、多操作系统版本、中文的排版和制作功能、对外设的控制、节目的包装与加密等。

第 7 章 网络基础知识

"计算机网络"常用的定义是：采用通信手段，将地理位置分散的、各自具备自主功能的若干台计算机有机地连接起来组成的一个复合系统，这个复合系统可用来实现通信交往、资源共享和协同工作等目标。本章将重点讨论网络的应用知识、软件知识与硬件知识。

7.1 网络的功能、分类与组成

7．1．1 计算机网络的功能

概括起来说，把若干台计算机组合成为一个现代计算机网络，可以实现以下三个基本功能：

- 计算机之间和计算机用户之间的相互通信与交往；
- 共享资源，包括硬件资源、软件资源和数据与信息资源；
- 计算机之间或计算机用户之间的协同工作。

计算机网络的通信交往、共享资源和协同工作三大功能，引申出使用计算机网络可有以下进一步的好处和用途：

1.通过构筑计算机网络产生一个性能/价格比更好的系统（如 Cluster 机群系统）

把计算机网络作为一个系统来看，合理的网络有可能节省整个系统建立、运行与维护的成本。在某些应用环境中，用一批 PC 机连成网络协同工作，与采用一个大型主机加上许多台终端相比较，在价格上更为便宜,对维护与运行人员的要求会更低一些。例如：世界杯足球赛等大型竞技运动会的管理通常会采用若干台大型主机加上一批 PC 客户机作为终端的方式运行，代价比较昂贵。1998 年底的曼谷亚运会采用的方案是：以 1800 台 PC 机和 200 台 PC 服务器，再用 140 个集线器和 60 台网络交换器连成一个完整的竞赛与管理场地的网络系统，这个亚运会信息系统沟通了新闻中心、国际转播中心、查询系统和因特网等，成功地形成了一个性能/价格比良好的运动会信息管理系统。

2．计算机网络可以提供具有更好可用性和可靠性的应用环境

在网络中从一台主机传输信息至另一台主机时允许选用多条路径,即使发生某一条路径的局部故障，仍能保证正常传输信息，不会引起整个系统瘫痪。当存放在某一台主机中的文件被破坏时,仍可以在网络的另一主机中找到可供使用的文件副本。特别是通过软件支持，当某一台主机有毛病时，网络中其他主机还可以去继续承担其处理任务,即整个系统可以在性能略为下降的状态下继续正常工作下去.在网上能够充分利用可替代的资源和备份的资源。

3． 在计算机网络内可以通过合理调度实现计算机之间工作负荷的均衡分配。

在网络中依靠软件的调度,可以把某段时间中工作负担特别重的主机中的部分任务分配给另外一些较空闲的主机上去执行,也可以事先就协调好网络中有关各台主机之间工作负荷的调派策略。在地域跨度很大的远程联网中，还可以充分利用"时差"来均衡主机的日夜负荷。

4．由计算机网络所构成的系统可以更方便地进行资源扩充和升级换代

当工作负担不断增加或者当系统的业务发生很大变化时，需要扩充和改变网络。在网络规模与性能扩充时，许多主机和网络设备不可以继续利用（或者降低其地位加以利用），在一定程序上保护了用户的投资。

5. 计算机网络的构成可以向用户提供更友善、更方便的用户使用界面和计算机资源的有效管理手段。

由于在计算机网络中面向用户使用的客户机（如 PC 机）与各类服务器主机可以在相应软件的支持下，分工完成相应的业务任务，因此有可能使最终用户通过图形用户界面（CUI）在 PC 机上专门从事面向用户的输入输出任务，比如：使用浏览器界面（也称为 Web，风格）方便地欣赏多媒体效果的输入输出。计算功能可由功能服务器或 WEB 服务器去承担,具体的数据库查询和加工则由数据库服务器

负责，用户只需极短时间的培训就能轻松地使用网络。另一方面，计算机网络上的有关软件能有效地控制与管理网络中资源的合理使用，只需网络管理员去设定用户权限、各类操作的许可权和各类资源的使用权约定或配置，网上用户就能各自依据自己所拥有的权限，科学地合法使用网络资源。

7.1.2 网络的分类

1. 按地理区域范围分类网络

（1） 局域网（local area network LAN）。典型应用场合为：

• 同一房间内的所有主机，覆盖距离 10 米的数量级。

• 同一楼宇内的所有主机，覆盖距离 100 米的数量级。

• 同一校园内、厂区内、院落内的所有主机，覆盖范围在 1 公里的数量级。在最后一种场合中，也常常称之为校园网。

（2） 城域网（netropolitan area network ，AMN）所有主机（工作站点）分布在同一城域内，覆盖范围约在 10 公里的数量级。

（3） 广城网（wide area netword ，WAN）所在主机与工作站点分布的地理范围能覆盖 10 公里、100 公里与 1000 公里以上的数量级，比如同一城市、同一国家、同一洲甚至跨越几个洲等。

（4） 互联网（internet）：也称为网际网，实质上是指两个或多个网络互相连接形成的（可以是非规整性的）网络。

局域网的基本特征是：

（1）整个 LAN 中所有物理设备分布在半径不超过几公里的有限地理范围之内(因此，不涉及到远程通信问题）。

（2）整个 LAN 由同一个组织或机构所拥有。

（3）在 LAN 中可实现相当高的数据传输速率，如传输速率范围可在 1Mbps-1000bps 之内（1M=100 万=2^{20}），常用 10Mbps-100Mbps。这里的 bps 是指每秒传送的二进位数，即 bit per second 的缩写。

（4） LAN 内的连接相当规整，有严格的标准可遵循.

广域网的基本特征是：

（1） 在 WAN 中信息的传输距离相对很长，可达几公里以上，甚至几千公里以上，涉及到对远程（非本地）计算机的存取与访问。

（2） 通常 WAN 分属于多个单位或部门所有，资源子网中的务类资源与通信子网中的名类资源与通信子网分别各自管辖与负责。比如通信子网一般由邮电管理部门管辖。

（3）在 WAN 的长距离通信线路上传输速率相对较低，一般在几十 K bps 至 2Mbps 的数量级左右，最高也不过几 Mbps 数量级（个别干线例外 ）。

（4） 网络的相互连接结构通常会很不规范，有相当大的随意性，甚至有一定有盲目性（很可能需要经过一个阶段作必要的调整）。

城域网是指介于 LAN 与 WAN 之间的一类网络，这是一类能覆盖整个城区甚至整个城市的网络(即传输距离在 10 公里的数量级).提出城域网概念是希望城区这样一个相对较大的地理范围内，提供数据、声音与图形等一些综合服务，要求其最高数据传输速率可达 50Mbps 左右。为了达到这个传输速率，MAN 主要采用了 LAN 的技术与标准。在设计 MAN 时曾提出过一种很有特点的访问控制机制，称为分布式队列双总线（distribudted queue dual bus ，DQDB）技术。

2．按信息传输技术分类网络

从在主机之间传输信息的基本技术或方式来看，可以大体上将网络区分为两大类：

（1） 广播式网络：在网络中只有单一的一个通信信道，由这个网络中所有的主机所共享。当从一个广播式网络中任何一台主机发送出一个短的报文（常常称之为一个"分组"）时，在网上所有的主机都可以接收到。通常在一个"分组"内的信息中有一个"地址字段（或者"标识符字段"）用来规定这个"分组"信息应当发送给哪一台主机（目的地）。每一台主机收到这个"分组"内容时，将会立即检查一下其中的"地址字段"。如果某主机发现发送来的是对应于它的地址的分组，那么该主机就要接收并且处理这个"分组"信息。如果发现不是它的地址，那么只要忽略之个分组就可以了。

这类广播式网络经常用于距离范围上，并且主机工作站点较少的计算机网络中，比如局域网基本上都采用广播式网络。

（2） 点到点网络：当在一个网络中成对的主机之间存在着若干对的相互联结关系时，便组成了一个点到点的网络。在每一对主机之间进行通信时，一台主机作为信息资源（发送地），另一台主机则作为信息的宿（目的地，这里的宿是指归宿的意思，即信息到达的终点或目的地）。允许一台主机可以与多台主机建立起成对通信关系。

点到点网络的一种常见组成方式在网络中许多对主机相互之间可以形成相对比较复杂的通路连接关系，如图 7.1 所示。

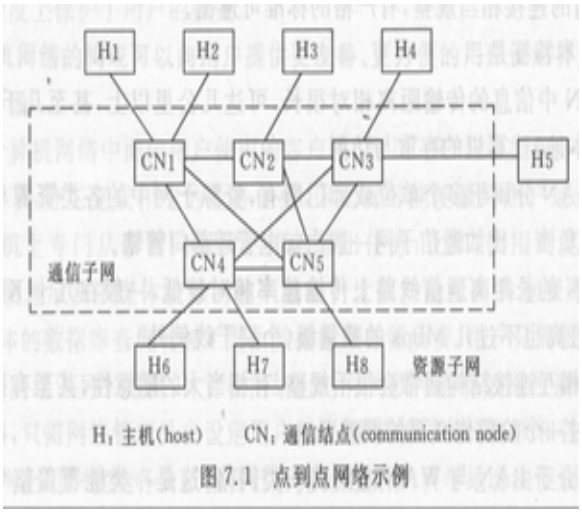


图 7.1 点到点网络示例

在图 7.1 中可见，从"源主机"向"目的地主机"发送一个"分组"信息时，这个"分组"信息可能会经由一个或多个中间通信结点才能到达目的地主机。比如：H1 主机向 H5 主机发送一个分组，可能经过 H1→ＣＮ１→ＣＮ４→ＣＮ３→Ｈ５这一路径。通常可以存在有多条可供选择的点到点通信路径，其中有最佳路径，也有较慢的路径。因此，路由选择的算法在这类点到点网络中起着重要的作用。

这类点到点网络方式通常用于较大的网络或者远程的网络，例如广域网基本上都采用了点到点网络的传输方式。

３．其他主要的网络分类方式

（１） 从网络传输信息所采用的物理信道来分类，可区分为有线网络与无线网络。目前较多采用的是有线网络方式组网。

（２） 从网络的使用范围来分类，可以划分为公用网和专用网两大类。公用网可为社会广大公众提供服务，只要愿遵守规定，进行注册并交纳少量费用，公众即可使用，比如：ＣＨＩＮＡＮＥＴ、上海热线等都理公用信息服务网。专用网则是为某个单位或部门、某个系统或某个行业等的特定业务需要而组建的网络，比如可细分为：金融信息网、指挥网、情报检索网、有许多专用网会有较强的保密性与安全性要求。

（３） 从网络的物理分布连接形状或拓扑构形来分类，可以把网络区公为星形连接网、环形网、总线网、树形连接网、完全连接风、交叉环形网以及不规则连接网。通常ＷＡＮ都是不规则形状的连接网。

7.1.3 网络的基本组成

任何计算机网络都由计算机硬件、软件、通信设备和通信线路(通信介质)所组成，此外，还应加上网络上有关的数据与信息内容。正如图 7.1 所示，也可以把一个计算机网络看成是由"资源子网"和"通信子网"所组成。

更具体地看，一个 LAN 大体上由以下三部分所组成：

(1)网络硬件：包括作为 LAN 站点的各台计算机及其配置的各种外围设备。如位于网内各台计算机中的网卡(一种插件式电路板)，可选的网络专用设备如集线器(hub)、交换机(switcher)和网络互连设备(如重发器、网桥等)，以及互连网卡与网络专用设施的传输介质(如：电缆或光纤以及连接转接插)等。

我们将在 7.3.1 中更祥细地讨论网卡、集线器、交换机、重发器等网络互连设施的功能和用途。通常把 LAN 中的计算机划分为服务器与客户机两大类。

(2)网络软件：通常是指实现 LAN 功能用的"服务器软件"和"客户机工作站软件"。一般说来，一个良好的"网络操作系统"软件已包括服务软件和客户机软件。当然，也允许使用其他的网络管理软件、通信软件等软件。

本章将在 7.2.3 节中祥细地介绍网络系统的功能和用途，在网络操作系统中将提供网络通信所必需的协议的有关功能。

(3)网络信息资源与网络应用程序：在网络上的数据与信息资源有可能涉及到相当广泛的内容，包括广义的"数据"(如声音、图像、视频文件、文档和各类数据库)，也包括应用程序、系统程序以及各类工具软件或其他软件。

如果把一个 LAN 看成由"资源子网"和"通信子网"所组成，那么 LAN 中的"通信子网"特别简单。主要是由传输介质、计算机的网卡和可选的网络连接专用设施(如 hub、交换机、重发器)等所组成。LAN 的"资源子网"则由 LAN 中的各台计算机(服务器与客户机)及其外部设备所组成，网络上的硬件资源、软件资源与信息资源都驻留在相应的各台计算机内。客户机面向最终用户工作使用，服务器作为资源的主要存放点和服务点。

一个 WAN 可以相当清晰地划分成由"资源子网"和"通信子网"两大部分组成。邮电部门或通信服务部门负责"通信子网"的组成、建设、管理与通信服务工作，在通信子网中可能包括有多台通信处理机(通信结点)、能信传输媒体(如电话线、光纤通信电缆等)和有关的其他通信设施，在相应的通信软件(协议实现)支持下完成通信服务工作。用户部门管理的"资源子网"中包含有一批主机或 LAN，其中的各台计算机分属于诸用户机构所拥有，WAN 中的所有用户资源都驻留在这些计算机中。

比如，即使用户是同一个大公司，在北京的总部要与广州的分公司进行远程联网，不可以也不允许该公司自行拉一根光缆线从北京连到广州。这个大公司只需要向邮电或通信部门提出申请，使位于北京与广州的 LAN(或主机)分别就地连入北京与广州的通信服务公网(公用网或专用网)的结点即可。有关通信子网运作与管理的全部事宜都由相应的邮电或通信服务部门全权负责，该公司只要支付相应的使用费用。

用户资源子网中的主机或 LAN 要可靠且稳定地连接邮电或通信服务部门的通信子网，需要有接入设备作为接口加以支持。接入设备可以是调制解调设备、路由

器或者是邮电与通信服务部门提供的专业通信接入装置等。我们将在 7.3.3 节中更祥细地介绍接入设备和接入技术。

7.2 网络协议和网络软件

所谓网络协议(有时也称为通信协议)指的是在计算机与计算机之间进行通信时，为了实现数 据有序的发送和接收，必须遵循的一些事先约定好的规则(标准或约定)，在这些规则中明确地规定了通信时数据的格式、数据传送的时序以及相应的控制信息与应答信号等内容。

为了保证计算机网络的开放性与兼容， 网络协议必须遵循标准化的体系结构，目前是指 ISO 的 OSI 标准或者 TCP/IP 协议组标准。

7.2.1 网络协议模型和 ISO/OSI 标准

1.网络协议层次模型的概念

在图 7.2 的左半部中，根据目前实际在使用的网络协议给出了一个网络协议的五层层次模型。

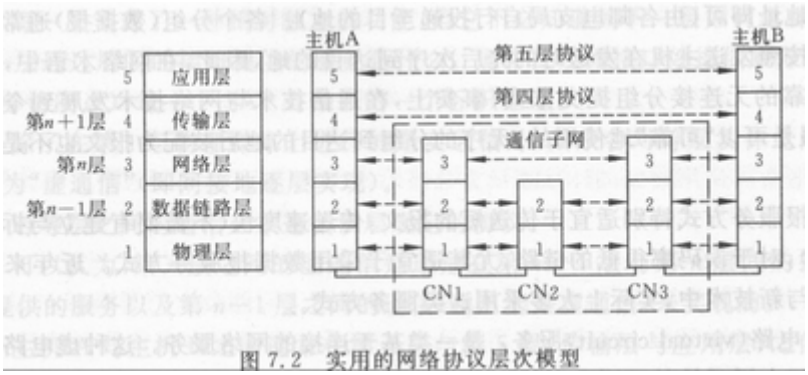


图 7.2 实用的网络协议层次模型

(1)物理层：给出了在一个通信信物的物理媒体上传输原始的二进位数据流(也称为比特流)时的协议。

比如：一个很常用的串行通信接口 EIA RS232 标准中详细规定了一个物理层协议。其中明确规定了通信双方的连接插头与插座的机械形状与要求，插针的数量与位置，每个插头引脚的信号涵义，信号的电气要求，电信号传输时电平上升、下降的时序控制关系，包括可用的信号连线和可以使用的物理介质等。从物理传输的角度，使通信双方有一个共同遵循的完整规则。

(2) 数据链路层：给出了把二进位数据流划分成为"数据帧",并依照一定规则传送与处理的协议.目标是把不够可靠的传输信道改造成为能可靠地传输帧的数据链路.这是因为在原始的二进位物理传输过程中,由于外界电磁干扰或噪声等原因,可能会引起传输内容的差错,当把二进位数据流重组成为数据帧时,规定了一个数据帧的哪些位是地址段,哪些位是控制段,以及哪些位是校验码.这样就有可

能采用校验接收到的内容、确认帧和重发帧等手段，保证正常接收到数据帧。分成数据帧后，还可以进行"流星控制"，防止发送方发送得太快，接收方接收过慢引起的数据丢失现象（称为"淹没"）。

（3）网络层：通过把需传送的数据划分成更小的"分组"（prcket），规定分组的格式，给出使分组经过通信子网正确地从源传送到目的地的协议。换句话说，网络层是控制通信子网正常运行的协议。

比如，分组交换网常用的 X.25 协议和因特网中采用的 IP 协议都是网络层协议的典型例子。在这些协议中规定了对数据"分组"格式，从源主机到目的地主机的路由选择、通信子网涉及的记帐处理以及当跨越不同通信子网时互联引起的不同协议、不同分组长度或不同寻址方式等问题的处置。在网络层通常可能会提供两类典型的数据分组传送服务方式：数据报服务与虚电路服务。

① 数据报（datagram）服务：也称为无连接的网络服务，这种"数据报服务"的工作方式雷同于邮电系统送信的工作方式，只需要在数据报（相当于一封信函）上有祥细目的地（收件人）地址即可（由各邮电支局自行投递至目的地）。各个分组（数据报）通常不必也不一定能够按照发送主机在发送时的先后次序到达目的地，因此，在网络术语中，称之为无序的不可靠的无连接分组提交方式（事实上，在通信技术与网络技术发展到今天的条件下，数据报是可以"可靠"地使用的，无序的分组到达目的地后装配为报文也不是一个困难的问题）。

数据报服务方式特别适宜于传送短的报文，传送速度快（不需要有建立与拆除链路的延迟时间），对于误码率很低的链路尤其适宜于采用数据报服务方式.近年来推出的许多新系统与新技术中，实际上大多采用这类服务方式。

② 虚电路（Virtual circuit）服务：是一类基本于连接的网络服务。这种虚电路服务方式十分类似于电话系统的通话，通话双方需先拨通电话建立"虚电路"，再允许通话，通话后挂断（拆除虚电路）。与电话系统中使用固定物理连接电路的交换方式不同的是，虚电路指的是动态地、临时性地、按照需要在逻辑上建立起多条可用的链路，然后逐个依次发送分组。比如我国邮电部门的分组交换网上已提供了几种虚电路服务。

虚电路服务的优点是：一旦建立起虚电路，"分组"信息可以在这条临时性的链路上按分组提交的次序依次传送，省去了报文中各分组的整序时间；也可以更加可靠而及时地发现传送中可能有的差错（特别是发现组的丢失）；一旦发现差错可以立即通知发送方及时重新发送出错的分组或丢失的分组以及以后的分组。

（4）传输层：也称为运输层。传输层协议可以根据高层用户的请求建立起有效的网络通信连接（事实上是通过网络层在通信子网中建立起网络连接路很），处理端到端（主机到主机）之间通信差错控制、恢复处理和流量控制问题，也可以方便地撤消与拆除网络连接。允许在必要时进行分流（即建立起多条网络连接来实现多个运输层通信）从而节省通信开支。

比如：因特网广泛采用的 TCP 协议是一个传输层协议，用来在一个不可靠的互联网上提供可靠的端到端字节流服务协议。这类传输层协议使主机用户不必了解通信子网的变化或结构，就可以方便地寻址主机、建立连接、释放连接、控制与调节流量和实现缓冲、多路利用网络连接、实施错误恢复、从而保证网络连接的服务质量。

（5）应用层：为各类不同的网络应用提供使用网络环境的手段，具体规定了在用户级别需要的、带有通信任务的许多常用信息服务的规则与协议。

在 7.2 的右半部中描绘了网络协议层次模型的基本工作方式：

（1） 第 n 层协议是第 n-1 层服务的用户，又是第 n+1 层的服务提供者。因此，在 n+1 层上的用户虽然看起来只是直接使用了第 n+1 层的服务，但实质上，还间接使用了第 n 层以及以下各层所提供的服务。

（2）每一层实现了一种相对独立的功能，但它不必知道其下一层如何具体实现，只要知道下一层通过层间的接口提供的服务是什么。通常把层间接口处提供服务的地方称为 SAP（服务访问点），每一个 SAP 有惟一的一个标识地址。

（3） 除了在第 1 层物理媒体上进行的是实通信外，其余各对等层实体之间进行的通信，都称为"虚通信"（即间接地逐层实现）。

（4） 对等层之间的通信必须遵循该层规定的协议。

（5） 可以认为，第 n 层之间的虚通信是通过在第 n-1 层与第 n 层间的接口处由第 n-1 层提供的服务以及第 n-1 层之间的通信（也可能是虚通信）来实现的。

（6） 主机 A 和主机 B 之间通常在第 4 层与第 5 层（传输层与应用层）之间实现虚通信。低三层的通信（网络层及以下层）可能经由通信子网进行。

（7） 通信子网中的通信结点 CN 仅涉及到低三层（甚至两层）的协议。

2. ISO OSI 网络协议层次模型

1983 年，国际标准化组织 ISO 提出的开放系统互联参考模型 OSI/RM 为网络协议的层次划分建立起了一个标准框架，在网络的发展史上发挥了杰出的作用。ISO OSI/RM 参考模型见图 7.3。



图 7.3 ISO OSI/RM 参考模型

在 OSI/RM 层次模型中，把网络协议规定成七层模型。即把上述图 7.2 中的第 5 层应用层进一步细分为：第 5 层会话层、第 6 层表示层和第 7 层应用层。

会话层（session layer）的作用是允许在不同主机上的各种进程之间进行会话，即在两台主机上的用户之间，建立会话连接，同步与控制会话（如：单向或双向传输）和结束会话连接。比如：在传输一个很长的大文件时，会话层可以在数据流之间插入一些同步点，当通信子网出现问题时，可从最近一次同步点开始重传以后的数据，这样就避免了在传输场合下，只能从头开始重新传输的做法。

表示层（presentation layer）为应用层提供传输的登记处在表示方面的规则与协议，即约定信息的语法（数据格式）与语义（信息解释规则、控制信息与错误处理等）规则及其表示时的各类变换规则。涉及到在不同主机上各种数据类开支和数据结构的表示方法和相到理解规则，通常需采用抽象的标准表示方法进行过渡。表示层协议也涉及到为数据的编码、加密与压缩等提供表示与变换功能。

在标准化的实践中，人们发现要把应用层归类或分解为 OSI/RM 所规定的会话层、表示层和应用层这三个相对独立的层次相当困难或者说可能会十分繁琐；另一方面，这三层的 ISO 标准制定工作也相对滞后，跟不上应用的需要。因此，在目前有效的衫网络协议中，事实上都把这三层含混地包括在应用层的协议内。

7．2．2 TCP/IP 协议组

TCP/IP 协议组的层次结构可参见图 7.4

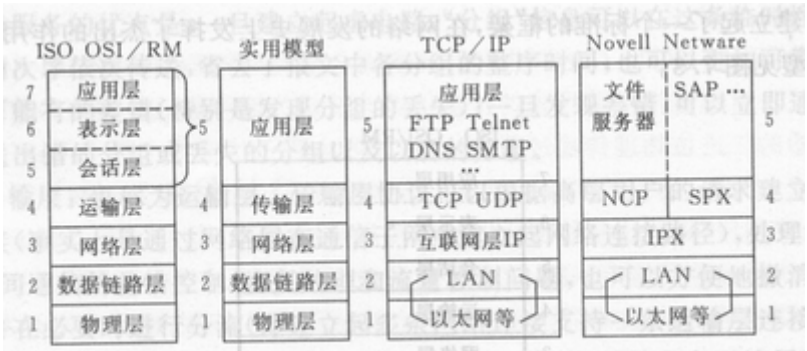


图 7.4 Novell Netware 基本结构与层次模型比较

1. 互联网层（即网络层）

用来方便地连接各种通信子网，负责路由选择合适的通信节点，使数据能从源主机发往目的地主机。
在互联网层有 IP 协议、ICMP 协议、ARP 协议与 RARP 协议。

（1） 协议规定了数据传输的基本单元（报文分组）以及所有数据在网际传递时的确切格式规范，定义了一组规则用于指定如何进行报文分组、不可靠数据的递交办法以及报文分组路由选择方法。显然，除了在主机上要采用 IP 协议外，用来接入通信子网的边界结点（路由器），通信子网中的各通信结点也需要使用 IP 层协议来实现数据的中继（接力式传递，存储-转发）。

（2） 网际控制报文协议（Internet COntrol Message Protocol，ICMP）主要用来供主机或路由器报告 IP 数据报在传输中可能性出现的不正常情况，比如：目的地不可到达、超时、分组首部参数有问题、重新定向路径等；也可通过应答（请求-响应）测试网络情况等，提供了一种控制报文传送的机制，可协助 IP 层实现控制。

（3） 地址转换协议（Address REsolution Protocol，ARP）用于完成 IP 地址向物理地址的转换.物理地址通常是指主机的网卡地址,每一网卡都有惟一的地址。

（4） 反向地址转换协议（Reverse address resolution protocol RARP）用于动态完成物理地址向 IP 地址的转换。
可以看出，IP 协议是互联网层的核心协议，负责 IP 分组的传送，ICMP、ARP 和 RARP 等协议仅提供辅助的控制功能。

2. 传输层

传输层的协议只存在于主机之中，提供主机之间进程与进程的有效数据传输。传输层使用端口号向应用层（传输层的上一层）中不同进程提供与该端口号相应服务。在传输层中有 TCP 协议与 UDP 协议。

（1） TCP 协议（transmission control protocol）：即传输控制协议，提供了可靠的面向连接的数据流传输服务规则和约定。TCP 协议是一个相对复杂一些的协议，因为它允许互联的网络中不同部分可能有着不同的构造路径的方法、不同的带宽、不同的延迟时间、不同的分组大小和其他不同的参数。TCP 协议的基本特点反映在以下几方面：

① 连接：为了获得 TCP 服务，数据的发送方主机与接收主机都要建立起一个端点，形成连接关系，这两个端点分别都称为"套接字"（socket），每一个套接字有一编号地址，事实上是由该主机的 IP 地址加上该主机酝规定的一个 16 位数据（"端口"号）所组成，一个"端口"（port）也就是一个传输层服务存储点（TSAP）。因此可以用两端的套接字标识号（Socket1，socket2）来标识"连接"关系。TCP 连接是双向同时可进行传输的，也是点到点的（每一次连接确切地对应两个端点）。传输数据后，必须释放和撤消连接。

② 数据流传输：每一台支持 TCP 的主机有一个 TCP 传输实体进程（如用户进程或者管理 TCP 流和交接 IP 层的内核的一部分）。该实体从本地各相应进程中接收用户的收据流，把它们分拆成小于 64K 字节的小片（大约定 1500 字节）并且作为一个个单独的 IP 数据报去发送每一片。当这些包含有 TCP 数据的 IP 数据报到达另一台主机后，交给相应的 TCP 实体，再重新构造成原来的字节流。

③ 可靠的数据传输：所有在 IP 层中未能实现而用户希望做到的可靠数据传输要求，TCP 必须予以实现。比如 IP 层不能保证数据报正常地递交，TCP 发现时间已到，根据需要重新传输；递交到达的数据报次序不正确，TCP 应根据正确顺序重新装配好。因此，根据上述要求在 TCP 协议中具体实现了数据段的格式规范、端到端连接的规则、传输数据时策略的约定、拥挤与阻塞控制的规则和定时定量管理的办法等。

（2） 用户数据报协议（User datagram protocol UDP）：该协议提供了无连接数据报服务的规则。这个无连接的传输层协议为应用提供了一种方式，允许发送封装好的原始的 IP 数据报，并且允许不必建立一次连接就可以发送这类数据。在许多应用中，只在有一次请示和一次响应之类的特点，就可以选用 UDP 协议。UDP 协议比 TCP 协议会有更好的性能和更高的效率（不必再考虑连接的建立与撤消），它也使用两个端口号和标识源目的地机器内的端点。

3. 应用层

提供了各种应用程序使用的协议，已有不少应用广泛的 TCP/IP 应用层协议，并正在不断发展。如 FTP、HTTP 等。

TCP/IP 协议组作为一种十分流行的网络体系结构，已成为事实上的工业标准。然而，TCP/IP 协议体系结构也有其不足之处，表现在下述各方面：

（1） 没有明显区分每一层中"服务"、"接口"与"协议"的概念，从而在使用新技术设计新网络时无法提供更有效的指导。

（2） 较难用 TCP/IP 协议模型（事实上这个模型是先有"协议"，再拼凑出来的）对应地去描述其他的协议栈或者新的协议。

（3） 对底层的物理层/数据链路层没有给出很好的定义，TCP/IP 也无法去区分。其他各层中"接口"与"层"之间的区分也太含混。

（4） 长期无次序的使用与发展也引起了一系列使用中的问题。比如：IP 地址的分配矛盾（目前有若干建议，如 IPv6）、IP 安全性、相对陈旧而贫乏的终端使用标准、图形用户界面设计的方便性等问题。

7．2．3 常用的网络操作系统

网络协议只是提供了网络通信双方的约定与规则，需要通过网络软件支具体实现网络协议中所规定的功能与规则。网络软件分布在主机中（服务器或客户机内），也位于通信结点和网络连接设备中。在主机中将驻留有最多层次的协议功能支持软件，在通信节点心或网络连接设备中一般最多只需要与网络层以下三层（甚至两层）协议功能有关的软件。

本小节讨论的网络操作系统是支持最多层次协议功能的软件，目前最流行的网络操作系统是 UNIX，Linux，WindowsNT 或 Windows2000sever，以及 Novell Netware。

1． 网络操作系统的基本功能

通常可以把网络操作系统定义为：实现网络通信的有关协议以及为网络中各类用户提供网络服务的软件的集合，其主要目标是使用户能通过网络上各个计算机站点去方便而高效地享用和管理网络上的各资源（数据与信息资源、硬件资源和软件资源）。

一个性能良好的网络操作系统，一般应用具备以下的一些特征：

（1） 允许在不同的网络硬件平台上安装与使用，运行各类网卡、多种传输媒体和实现各类常见的媒体访问控制协议和物理层协议。

（2）能向各类用户提供友善方便而高效的用户界面，便于使用与管理网络，也便于地配置与利用资源，并且应当有较迅速的服务请求响应能力.

(3)能支持常见的各类"多用户环境"的应用,也可支持用户的协同工作.

(4)能有效地实施系统安全性保护和存取权限控制.

(5)提供有网络资源控制与网络管理的各类实用程序或工具软件,比如:经常需要用到的系统备份、性能监测、配置修改和安全审计或防范等。

（6）提供必要的网络互联支持（比如，当在网络中采用不同协议的网卡时，必须提供网桥支持功能）。

2. Novell Netware

Novell 公司推出了 Netware3.XX 版本以后，就占领了以文件服务与打印服务为主要目标的 LAN 应用的大部分市场。当 LAN 使用了 Novell Netware 作为网络操作系统时，即被称为 Novell 网。Novell 网的文件服务与目录服务功能相当出色。Netware 的基本结构可用图 7.4 右部形象地说明。图 7.4 也给出了 Novell 网、TCP/IP 体系结构与本章上面所述的所有实用模型层次结构之间大致对照关系。

第 3 层网络层采用的是 IPX 协议，这是一类不可靠的无连接的网络互联协议，这个协议的功能是把 IPX 格式的"分组"（包）数据透明地从源送到目的地（允许源主机与目的地主机位于不同的网络）。IPX 的功能十分类似 TCP/IP 中的 IP 协议（但采用了 10 个字节的地址，而不是 IP 地址用四个字节），采用互联网数据报分组方式工作。

Netware 在运输层有两个主要协议 NCP 和 SPX。NCP 称为网络核心协议（netware core Protocol），这是一个面向连接的传输层协议，是 Netware 的核心部分，除了用户数据传输外还提供有各种其他服务。在运输层也可使用 SPX 协议，但 SPX 协议仅提供传输层功能，另一个要选协议即是 TCP 协议。一般的网络应用可以选用其中的一个协议（比如：文件系统常用 NCP，而 Lotus Notes 应用软件会使用 SPX 协议）。

各类应用协议出现在应用层，如文件服务器中的一些支持协议，在 LAN 中通常采用广播式通信，比如大约一分钟一次由每一个服务器去广播一个分组，在分组信息中给出地址和它所提供的服务。这类广播使用的是 SPA 协议（service advertising protocol，服务广告协议），客户也使用广播方式请求服务。

为了保证实现多用户共享资源的功能，Netware 采用三层结构的软件实现方式：

• 最内层是系统核心调度程序，这是实现多用户的基础层。

• 次内层是一批面向用户的管理程序，包括磁盘管理、打印管理、网络通信管理与控制台命令处理程序等。

• 最外层是所谓的 shell 外壳，用来分析解释各站点提出的服务请求。
从安装使用角度看，Netware 网络操作系统事实上是由文件服务器网软件、客户机联网软件、可选的网桥内互连软件以及可选的网络增值服务软件等所组成。如：客户机联网使用所谓的"工作站外壳"软件 NETX（低版本）或 VIM（高版本）。

3. Microsoft Windows NT Server(或 Windows 2000 Server)

NT 主要面向应用处理的领域，具有多任务与多线程机制支持，特别适合于客户机/服务器方式的应用。近年来还将因特网信息服务与分布式数据库存支持等功能集成到 NT Server 之中，目前已可广泛地用于数据库服务器、信息服务器、部门级应用服务器、企业级应用服务器、通信服务器、文件与打印服务器和事务处

理服务器等各类应用场合。从 ISO/OSI 模型的观点来看，NT 把网络层、运输层和会话层结合成为一个"网络运输系统"。

事实上，目前网络层与运输支（及会话层）最常用的是有三大类协议：TCP/IP，Net BIOSt 和 IPX/SPX 兼容协议。当涉及因特网或网络的远程互联时，目前已多采用 TCP/IP 协议。我们 已看到 Novell 采用的是 IPX/SPX 协议，当 NT 使用 IPX/SPX 兼容协议时，可以支持任何与 Novell Netware 兼容的网络用户组件，可以把客户机工作站与 Netware 服务器以及 Windows NT 服务器连接起来，共享网络资源。

NT 默认安装的 NET BEUI 协议，是由 IB 公司于 1985 年以后推出的协议，即是"Net BIOS 扩展用户接口"的意思，此协议是用约 20-200 台客户机规模的中小型 LAN 设计的，具有很强的流控制能力，并且提供了较强的参数设置和差错检测功能。对无需跨越路由器与大型主机通信的中小型 LAN 来说，有了 NET BEUI 后可以不必再安装其他协议。微软在其所有网络软件（包括 NT 和 LANMANGAGER 在内）中都采纳了 NETBEUI 作为基本的网络层及以上层的网络层及以上支的网络传输协议，也允许选用 TCP/IP 协议。NT 还可以选用微软自己开发的 DLC 数据链路控制协议，可用同一个网卡建立起与各种 IBM 主极及 AS/400 计算机的连接，也可以用业提供与直接连在网卡建立起与各种 IBM 主机及 AS/400 计算机的连接，也可用来提供与直接连在网上的本地打印机的连接。

从 NT 的内部系统结构组成来看，可以把 NT 划分成为 NT 保护系统和 NT 执行体两大部分：

(1)NT 保护系统也称为系统用户状态部分，采用了客户机/服务器模型（详见以下的 7.5.2），它所有关客户端进程与服务器进程组成，相当于 NT 的最高层。

（2）NT 执行体也称为系统核心状态部分，这又是由一组组成部件所构成的一个四层的导次结构。其中的最外层是系统服务（为 NT 保护系统的进程所提供的接口）；次外层是由系统服务下的几个部件所组成（如本地过程调用功能、虚存管理、进程管理对、对象管理、安全调用监视等）允许采用内部界面去相互调用；次内层侧是由系统服务下的几个部件所组成(如本地过程用功能、虚存管理、进程管理、对象管理、安全调用监视等)，允许采用内部界面去相互调用；次内层则是当小的一个"内核"；最内层称为硬件抽象层，使执行体的其他部分能与硬件特性相分离。

WINDOWS 分为 NTSERVER(安装在服务器上的网络操作系统)和 NTWORKSTATION(可供安装在客户端机器上的联网软件，也是一个很好的客户端本地操作系统）。在服务器上安装了 NTSERVER 后，该网络可称为 NT 网格。也允许在 NT 网络的客户端上安装 WINDOWS98（WINDOWS95）基至 WINDOWS FOR WORKGROUP3.11 这时客户端可以降低一些配置（如内存量和硬盘容量）。

4、UNIX

由于历史原因，UNIX 系统平台目前仍是运行 TCP/IP 协议的首选平台，能够较好地提供系统的互操作性，应用的可移植性和互联网络的有效支持。比如因特中较大的服务器几乎无一例外地采用 UNIX 操作系统。目前 LINUX 应用推广相当迅速。

从网络应用功能来看，UNIX 系统在上层功能中重点实现以下几个方面的内容：

（1）文件管理，包括文件复制（如 UUCP 实用程序把非本地文件复制到本文件系统）联合文件系统（允许在用户程序中对非本地文件实施一些常规文件操作）和保护文件（包括对非本地与本地文件的存取权限控制检与检查、自动映照用户的标识等）。

（2）在网上管理用户程序的分布与执行。

（3）网络内点到点的邮件传送（E-mail）和文件传输（FTP）

（4）网络内非本地的打印输出服务（因为邮件与打印文本等均以文件形式组织。许多应用都可采用"文件复制技术"为基础进行。）

在 UNIX 系统中提供了一批 TCP/IP 常用令。比如：

ping：检查互联网中两台主机之间是否可正常地连接通信。

telnet：用户仿真终端可远程登录到其他主机上，使远程主机执行相应的命令，并返回结果。

ftp：在本地与远程主机之间传送文件。

finger：显示本地主机或远程主机上用户的有关信息。

ruptime 和 rwho：分别显示本地网络上所有主机信息和当前登录的用户信息。

talk：允许互联网上两个用户进行实时时对话。

mail：进入电子邮件系统。

特别值得指出的是，SUN 公司是 UNIX 平台标准化和网络文件系统的积极推进者之一，它 1984 年所推出的 NFS（网络文件系统或网络文件服务）协议，在 UNIS 系统上推进了 LAN 文件传输、文件存取与分页等功能的标准化。这种较好地实现共享远程文件地协议，已成为事实上的工业标准。NFS 的下部层次采用 TCP / IP，用远程过程调用（RPC）和外部数据表示协议（XDR）保证了 NFS 应用的互操作性。

UNIX 系统作为 TCP / IP 协议优选平台的主要原因在于：UNIX 开发与应用的历史较长（远早于 PC 机的出现），在长期实现 TCP / IP 协议功能的过程中相对更加开放和更易于标准化，实现互操作性的代价相对较小，UNIX 在本质上可有效地支持多任务与多用户工作，设计时就有面向集中管理和保护安全性的特征，适宜于在 RISC 等高性能主机上安装运行。这些都使 UNIX 有着很强的生命力。

7.3 构建网络与申请通信服务

本节将重点介绍网络硬件的基础知识，无论在哪一个单位或部门，增添网络硬件的任务主要体现在构建与配置 LAN 的过程中，7．3．1 中将加以讨论分析。相对而言，在建立 WAN 时则应依据本单位 LAN 的具体情况，结合申请通信服务和使用网络接人设备，对 WAN 进行必要的规划就可以了（在 7．3．2 节～7．3．4 节分别予以介绍）。

7．3．1 构建 LAN

1．LAN 涉及的主要网络专用设备

（1）网卡：也称为网络适配器或者网络接口卡（network interface card，NIC）。网卡及其驱动程序事实上已基本实现了网络协议中底部两层的功能。具体负责主机向媒体收 / 发信号，实现帧一级协议的有关功能。

（2）集线器：HUB 的英文原意是电器控制面板上的电线插孔座集中地，从这里可接出传送信号的引线。在网络中把 HUB 称为集线器，也称为线集中器。把站点都集中到"总线"上相互连接在一起，也可以使 HUB 之间相互用双绞线进一步互联接通。比如：可以先把每间小房间里的计算机先连接在相应的一个 HUB 上，再把这些 HUB 互相连接而构成一个 LAN 网络（见图 7．6）。

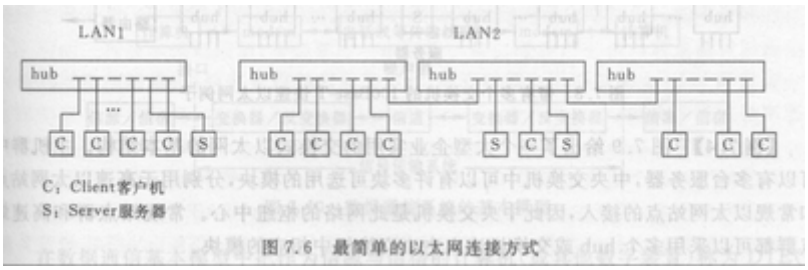


图 7.6 最简单的以太网连接方式

HUB 也可分为共享式 HUB、堆栈式 HUB 和交换式 HUB。共享式 HUB 和堆栈式 HUB 整体作为一个网段；而交换式 HUB 的每一个口都允许作为一个网段，速度非常快。

（3）重发器：也称为中继器或转发器，这是一种在物理层上互联网段的小设备。因为信号在传输媒体的线路上传输一段距离后必然会发生衰减或者畸变，通过重发器放大增强信号并进行转发就可以保证信号可靠传输。采用重发器把两条（或更多条）干线连接起来的方法，使这两个干线段成为同一个 LAN。

必须注意的是，重发器连接的两个网段，必须是同一种类型的 LAN。上段提到的 HUB 事实上可起到多端口的中继器（重发器）的作用，即允许将多个网段连到同一 HUB。

（4）网桥：也称为桥接器（bridge）。这是一种在数据链路层把同段互相连接起来的设备。在网桥中可以进行两个网段之间的数据链路层的协议转换。

从概念上讲，网桥最重要的功能是对数据进行"过滤"的能力。即在网桥中保存着所连接的每个网段上所有站点的地址。当收到一个帧时，可以只让必要的数据信息通过网桥或者只向相应的网段转发。比如：如果网桥收到的帧中的目的地址与源地址属于同一个 LAN 网段，则只把它卸下来；如分属于两个 LAN 网段，那么可进行帧格式转换后继续向前转发，这时也过滤下了帧内的有用信息而产生出新的帧格式。网桥也可以把坏帧与校验出错的帧（或分组）过滤掉，使之不再扩散。

网桥的主要参数有：网桥的过滤能力（以每秒过滤多少个分组 PPS 来衡量），是否允许用户定义过滤规则，网桥的转发速率以及网桥是否具有自学习能力（所谓自学习能力是指当信息分组或帧通过网桥时可以自动地构成一个表，用来确定与调整用户处在哪一个网络位置，这个表称为过滤表）。

（5）交换机：也称为交换器。一台基本功能的以太网交换机的工作原理相当于一个具有很多个端口的"多端口网桥"，即是一种在 LAN 中互连多个网段，并可进行数据链路层和物理层协议转换的网络互联设备（见图 7．7 和图 7.8）。

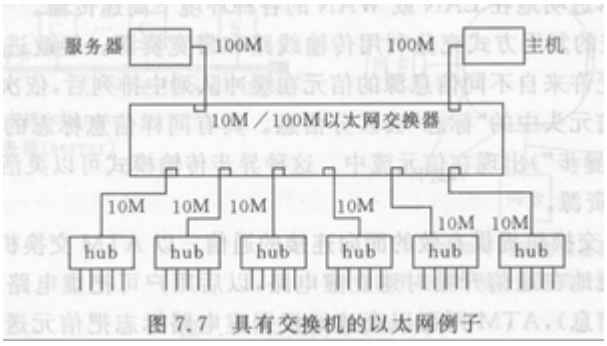


图 7.7 具有交换机的以太网例子

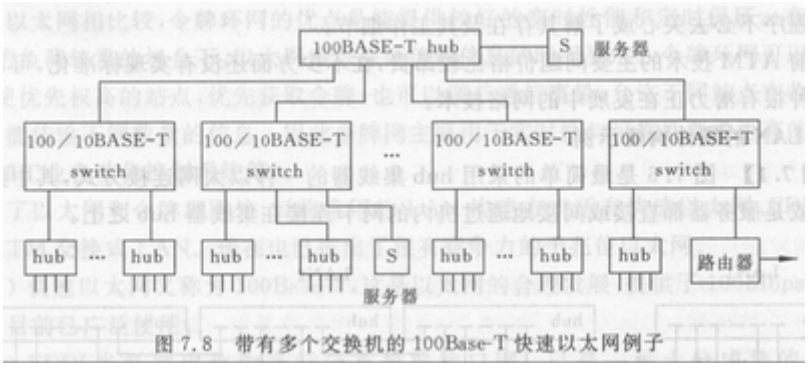


图 7.8 带有多个交换机的 100Base-T 快速以太网例子

当一个以太网的信息帧到达交换机的一个端口时，交换机根据在该帧内的目的地址，采用"快速技术"把该帧迅速地转发到另一个相应的端口（相应的主机或网段）。交换机这种高速地在端口间切换帧（或分组）的特征，保证了每一网段或每一台计算机可实现在网络内的高速信息交换。多个端口可以提供并行地使所有网段或计算机同时全速传递与交换信息的能力。
目前在以太网交换机中最常用的高速切换技术有两类：

（1）直通式（cut－through）：当信息帧到达端口时，交换机读出帧内目的地址后，立即查询在其内部 RAM 中保存的端口与站点地址对照表，如果有某站点地址相符合，随即把帧转发至相应端口；如果没有任何对应则忽略该信息帧，不进行任何错误校验。

（2）存储转发式（stored-forwarding）：类似地，当信息帧到达端口并读出帧内目的地地址后，如果与某站点地址相符，则立即把整个帧全部存人交换机的″高速缓存″内，在进行错误校验后，再把帧转发至站点相应的端口。显然，存储转发式交换机有更好的可靠性,并可匹配不同的传输速率;但比直通式要慢一些。

2．网络采用的主要传输媒体

目前在网络中数据与信息传输的(有线信道)线路主要采用以下三大类传输媒体：双绞线、同轴电缆与光纤。

(1)双绞线：这是使铜质导线相互之间按一定扭距绞合在一起的一种传输媒体，十分类似于室内电话线。双绞线可按其是否外加金属网丝套的屏蔽层而区分为屏蔽双绞线（STP）和非屏蔽双绞线（UTP）。电气工业协会／电信工业协会（EIA／TIA）把双绞线按其电气特性加以分类网络中最常用的是三类线和五类线，目前已有六类以上的线。

第一类双绞线事实上是电话线，用于模拟话音的传输，在计算机网络中不使用。第二类双绞线可用于数字话音、IBM3270 或 N－ISDN（窄带综合业务数据网）低速传输，在 LAN 中也不使用。第三类双绞线在 LAN 中常用作为 10MbPS 以太网的数据与话音传输，符合 IEEE802．3 10Base－T 的标准。常见规格是 24WG 的四对非屏蔽双绞线 UTP（符合 EIA／TIA568 标准规定的 100Ω 水平布线电缆）。第四类双绞线性能优于第三类，可用于包括 16MbpS 令牌环网在内的数据传输（可以是 STP 或 UTP）。第五类双绞线目前占有最大的 LAN 市场，最高传输速率可达 100MbPS，一类常见规格是 24AWG 的四对非屏蔽双绞线。与三类线一样，五类线也采用 RJ45 夹头作为插头接人网卡或 hub 等网络设备的 RJ45 插座内。RJ45 插座也相应地分为三类或五类电气特性。

为了适应高速传输数据的需要，正在推广与制定六类（200M）、七类（600M）等更高性能的双绞线标准。

（2）同轴电缆：在 LAN 中主要采用细缆（适用于短距离、较少站点）和粗缆（适用于较长距离或较多站点）。最常用的同轴电缆有：细缆--RG－58 用于以太网；粗缆--RG－8 或 RG－n 也用于以太网;有线电视电缆--RG－59 可用于有线电视信号传输（75Ω 阻抗），也可用于宽带数据传输。

（3）光纤：光纤由单根玻璃纤维纤心、纤心的包层和塑料保护涂层所组成。光纤的两端必须配备有光发射器件或设备、光接收器件或设备才能传输信号。通常用 LED（发光二极管）和 ILD（注人式激光二极管）作为电光转换器件，用光电三极管或二极管作为光电转换器件。

根据光在光纤中的传播方式，当前有两大类常用的光纤：

①单模光纤是价格最贵、性能最好的光纤。通常需采用 ILD 作为发光器件，传输速率可达几个 GbPS 的数量级，传输距离可达 30 公里。单模光纤的纤心直径小到 8～10μm 以下，即只有光波的几个波长，类似于波导，从而使光只能以直线方向传播，通过光纤接收到的信号与输人信号已大体相同。

②多模光纤，其中多模渐变型光纤的信号保真度较好，价格较便宜；多模突变光纤的信号散射相对严重一点，是价格最便宜的一种光纤。多模光纤的纤心直径大约在 50μm 以下（相当人的头发丝），这时发光管发出的光线中凡是以小于临界角所发射的那些光射线，都在光缆包层界面发生反射，以多次内部反射的方式沿纤心传播。把这些以不同角度反射的光射线称为具有不同"模式"的传播，所以称为"多模"光纤。多模光纤也能以几百 MbPS 的速率传输信息。

3. LAN 的协议标准和主要的媒体访问控制方式

在 LAN 标准中主要关心物理层与数据链路层的网络标准.把数据链路层又分为媒体访问控制子层 MAC（与不同的媒体有关）和逻辑链路子层 LLC（与媒体及其访问方式无关）。目前 LAN 已有下列 11 个主要协议标准。该标准正在扩展之中，至 2001 年已达到 16 个，统称为 ISO 8802 标准（或 IEEE 802 标准）. 包括:

ISO 8802.1--体系结构（参考模型）、管理与寻址（各标准的关系）、网络互联系统；
ISO 8802.2--LLC 子层标准（包括连接方式、信息结构与主要功能等）；
ISO 8802.3--以太网（CSMA / CD 总线网）；
ISO 8802.4--令牌总线网；
ISO 8802.5--令牌环网；
ISO 8802.6--城域网；
ISO 8802.7--宽带 LAN；
ISO 8802.8--FDDI（光纤局域网）；
ISO 8802.9--ISDN 局域网；
ISO 8802.10--LAN 的安全性管理；
ISO 8802.11--无线局域网。

已经推出与制定了千兆位以太网（802. 3z 和 802. 3ah）等有关标准。8802. 3～8802. 11 标准中的大多数都规定了相应媒体的物理层协议和 MAC 子层协议标准。

当前使用最广泛的 LAN 是以太网，令牌环网也有一些应用。分别介绍如下：

（1）以太网（Ethernet）：如图 7. 5 左部所示，也称为 CSMA / CD 总线网。其主要特征是所有 LAN 中的站点本质上都连接在一条公共的总线上以广播方式通信，采用带有冲突检测（CD）的载波侦听多路访问（CSMA）的媒体访问控制原理进行工作。CSMA 指先侦听，到发现总线空闲时再发送。CD 指开始发送时，也检测冲突，发生冲突时等候一段时间再用 CSMA 方式发送。常规的以太网能以 10MbpS 作。

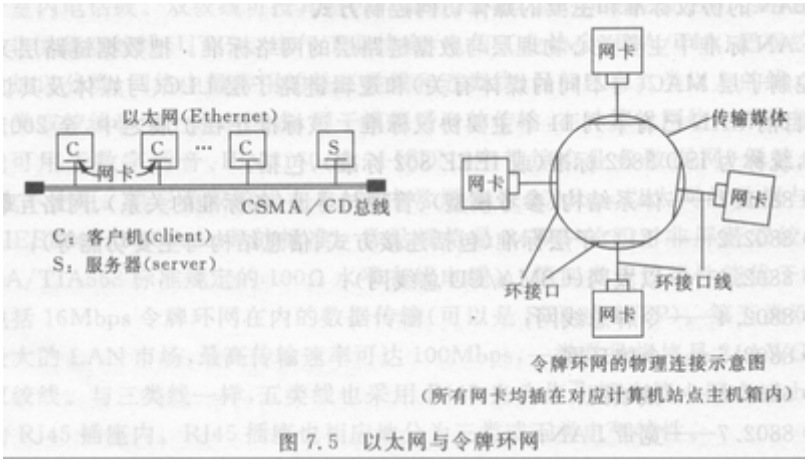


图 7.5 以太网与令牌环网

（2）令牌环网（token－ring）：如图 7．5 右部所示。这是由一组连接成环形的计算机站点组成的 LAN，在各个站点之间依照在环上的次序传输令牌和帧信息。所谓"令牌"事实上是发送信息的一种许可证，指的是一个特定的信息帧，可以在环上逐个站点地流转传递，告诉希望发送信息的各个站点是否有权可立即发送信息。常规的令牌环网最高可用 16MbpS 工作。

令牌环网的基本工作原理是使令牌不断地在环上依次向各站点传递，只有获得了"空闲令牌"的站点才有权向环上发送信息（向下游站传递）。当环上所有站点空闲时，令牌称为空闲令牌，当一个站点无信息传送时，只需简单地让令牌依次传向下一站点。某个希望传送帧的站点必须等待，直到它检测有一个空闲令牌经过时，才获得发送机会；这时它把空闲令牌改为忙令牌，并且紧接着忙令牌后传出一个帧(分组)信息。接收信息的某个站点可以复制通过环发送给它的信息。当忙令牌循环一周返回到原发送站点时，原发送站在在环上加一个新的空闲令牌，并清除环上原发送的帧信息。

与以太网相比较，令牌环网的优点是能提供较好的实时性能和定时保证。在站点较多、通信负荷较重的场合下，以太网难以保证传输信息流量，允许不同站点在收到空闲令牌时能传输不同数量的信息。因此令牌网主要用于实时性与可靠性要求更高的一些应用中(如工业自动化控制系列等)。

除了以太网和令牌环以外，目前常用的 LAN 构建还有快速以太多、FDDI 光纤网和 ATM 交换式 LAN。现在也已推出了很有竞争力的千兆位以太网。

(1)快速以太网又称为 100Base-T,这是以太网的合理发展，提供了 100Mbps 的传输速率。目前已广泛使用。

(2)FDDI 光纤网也称为光纤分布数据接口网。已是一种十分成熟的可提供 100Mbps 传输速率的标准化技术。采用了一种双环路结构，实现经改进后的定时令牌环网的访问机制，允许有多个数据帧同时在环路上传输。但是 FDDI 网价格较高，主要在一些单位中用作 LAN 的主干网。

(3)ATM(异步传输模式)是一种迅速崛起而倍数关注的交换技术，仅以 53 个字节固定长度的信元作为基本传输单元，可以实现 155Mbps、622Mbps 甚至更高的传

输速率。目前已在一些企业集团、重点大学和一些重要机构等作为主干网，成为校园网和企业网提供信息交换的高速主干通道。ATM 的基本特征包括：

①提供传输应用的多样性。在 53 个字节的信息元中，其中包括控制信息；48 字节是信息段(也称有效负荷)，其中可包含来自于不同业务的用户信息(任何类型的信息)。所有各类信息(包括语言、图像、视频、数据等)都划分成信元后采用可伸缩的单一化通信方式，透明地在 LAN 或 WAN 的各种环境下高速传输。

②采用经改造的复用方式充分利用传输线路和带宽资源。经改进的复用也称为异步时分多路复用，允许来自不同信息源的信元在缓冲是排列后，依次在传输线路加形成"信元流"，依靠信元中的"标志"去区分信息。具有同样信息标志的各个信元允许随机且非周期性地("异步")出现在信元流中。这种异步传输模式可以灵活并最大限度地充分利用 ATM 网络资源。

③通过 ATM 交换机提供有效的面向连接的通信。以 ATM 交换机或 ATM 交换模块为核心，可以方便地在通信开始时建立虚电路，以后用户可把虚电路标记写入信元头(相当于逻辑地址信息)，AMT 网可以高速地根据虚空电路标志把信元送往各目的地。交换机采用高速硬件直接完成面向连接的虚电路交换工作，以保持信元发送资源的发送的次序的方式取得高速效带效果，可以匹配于光纤的高速传输速率。

④已经和正在发展若干技术，允许一个 ATM 交换主干网能通过路由器或 hub 把 LAN 与 WAN 透明地连接起来，向用户提供透明的传输服务，这里的"透明"是指相应的用户或程序不必去关心或了解其存在及其工作细节。

目前 ATM 主要问题价格比较昂贵，在不少方面还没有实现标准化，可以认为这是一种很有潜力正在发展中的网络技术。

4.LAN 的硬件构建示例

〖例 7.1〗图 7.6 是最简单采用 hub 集线器的一种以太网连接方式，其中无论是客户机或是服务器都直接或间接地通过的机内的网卡连接在集线器 hub 之上。

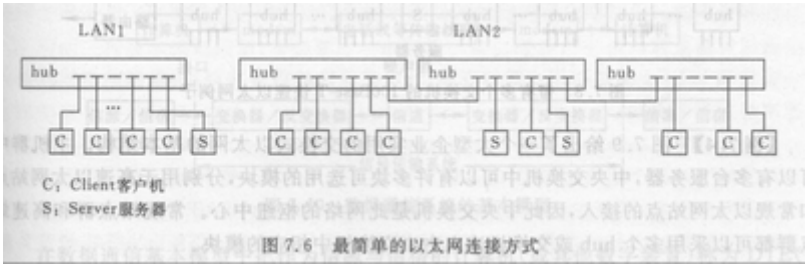


图 7.6 最简单的以太网连接方式

其中每台计算机的网卡都采用双绞线与RJ45插头座与hub连接(也可使用细缆或粗缆作为连接线路的媒体)。

〖例 7.2〗图 7.7 是具有交换机的以太网示例，其中采用了一台 10M/100M 以太网交换机，客户机共享 100M 带宽的服务器。每台 hub 上连着 5 台客户机，可以共享 10M 带宽。交换机具有 6 个 10M 端口和 2 个 100M 端口，是一种小型的交换机。

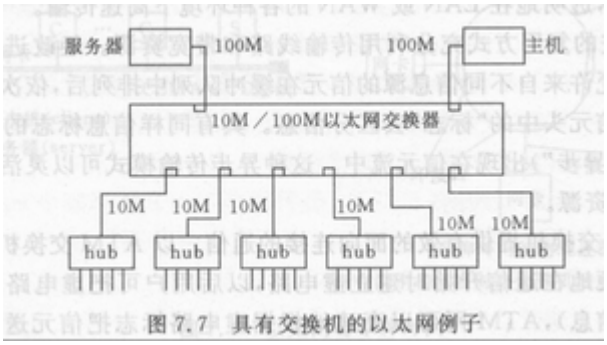


图 7.7 具有交换机的以太网例子

〖例 7.3〗图 7.8 是具有多台交换机的一个中等规模的以太网示例， 顶部的 100Base-T hub 也可以是一台交换机(交换式 hub)。右下角的路由器用来连接其他网络。这是一个很典型的 100Base-T 高速以太网的例子。

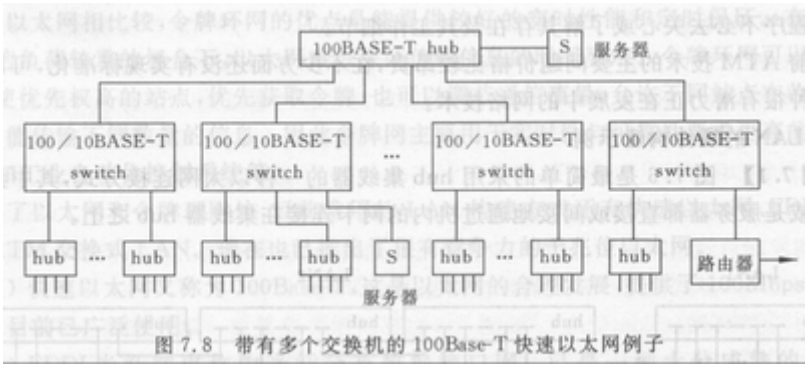


图 7.8 带有多个交换机的 100Base-T 快速以太网例子

〖7.4〗图 7.9 给出了一个大型企业实用的交换式以太网的基本结构。主机群中可以有多台服务器，中央交换机中可以有许多块可选用的模块，分别用于高速以太网站点和常规以太网站点的接入，因此中央交换机是此网络的枢纽中心。常规站点群和高速站点群都可以采用多个 hub 或交换机连入中央交换机中相应的模块。

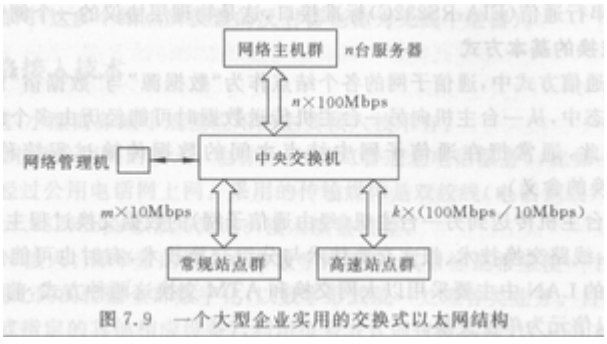


图 7.9 一个大型企业实用的交换式以太网结构

7.3.2 申请通信服务

在构建 WAN 广域网时，必须考虑向邮电部门或 ISP(因特网服务提供商)申请通信服务的问题。本节将讨论数据通信、数据交换方式和邮电部门提供的通信服务。

1.数据通信的基本概念

计算机数据通信指的是，通过通信信道在计算机之间数据与信息的收集、传输、交换或重新分布的一个过程。数据通信涉及到以数字形式或者模拟形式发送数据与接收数据这两个主要环节。通常把发送数据的来源称为信源或数据源，而把接收数据的目的地称为信宿或数据宿。

例如一台作为信源的计算机与一台作为信宿的计算机经过电话网通信时，需要在信源发送处插入调制解调器(modem)把数字信号调制为模拟信号，再在电话线路上传输，然后在接收点又 用 modem 把模拟信号解调为数字信号，由信宿计算机接收。如图 7.10 上部所示。对照之下就有图 7.10 下部的数据通信系统的基本模型。

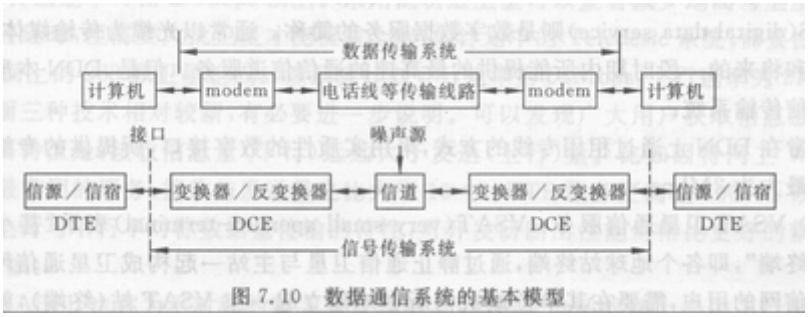


图 7.10 数据通信系统的基本模型

在数据通信基本模型中把作为信源与信宿的计算机(或其他数字装置)称为 DTE(数据终端设备)，把变换器(如调制器)和反变换器(如解调器)称为 DCE(数据通信设备或数据电路端接设备)。在 DTE 与 DCE 之间应当有标准的接口，比如 MODEN 与 PC 计算机常用 COM 串行通信(EIA-RS232C)标准接口，这是物理层协议的一个例子。

2.数据交换的基本方式
在点到点通信子网的各个结点作为"数据源"与"数据宿"的地位是处在动态改变的状态中，从一台主机向另一台主机传送数据时可能经历由多个结点所组成的一条路径。因此，通常把在通信子网中结点之间的数据传输过程统称为数据交换(switch，有切换的含义)。

数据从一台主机传送到另一台主机(经由通信子网)的数据交换过程主要可选用一种数据交换技术：线路交换技术与分组交换技术，有时也可能组合其中的某些技术。目前的 LAN 中主要采用以太网交换这两种方式，前者以分组为单位交换，后者以信元为单位变换，

3.邮电部门提供的通信服务：

(1)普通电话网计算机通信服务：任何一个家庭或任何一上单位中的 PC 机只需加上 MODEN，再加上一条程控电话线就可以使用公用普通电话来连接对方(可以是家庭、单位或任何"服务提供点"，也可以是邮电部门的公共数据网节点)，要求对方也具有 modem 加接主机。此外，计算机内必须装有通信软件。采用是以拨号呼叫的方式使用通信子网(现在是公用电话网)，实现远程通信。

(2)分组交换公共数据网服务(CHINAPAC)：目前的 CHINAPAC 可提供的最高传输速率为 64kbps，可以比普通电话网有高得多的信道容量和可靠性。当前，我国邮电部门在公组交换网(CHINAPAC 或本地，例如上海市的 SHPAC)上产要提供两种基本服务业务，即"交换虚电路"服务与"永久虚电路"服务。采用 X.25 协议，称为 X.25 分组交换网。

(3)数字数据网服务(DDN，DDS)：DDN 是数字数据网(digital data network)的简称，DDS(digital data service)则是数字数据服务的简称。通常以光缆为传输媒体，这也许是目前和将来的一段时期内所能提供的最高速的通信信道服务。但是 DDN 本质上只是一个通信传输系统。

通常在 DDN 上通过租用专线的方式，采用实质性的数字接口，所提供的专线线路传输率量在为 2MBPS。

(4)VSAT 卫星通信服务：VSAT(very small aperture-terminal)表示"甚小口径的地球站终端。即各个地球站终端，通过静止通信卫星与主站一起构成卫星通信网。作为卫星通令网的用户，需要在其单位或机构所在地建立起一座 VSAT 站(终端)，就可以通过卫星进行终端站之间的数据、话音等的各项信息通信。通常，VSAT 站所需的设备简单，组网灵活。当由地面发射的一个电波通过静止通信卫星再到达地面站接收，这个行程(称为传送一个电波反射)约需要延迟 0.27 秒，这是卫星通信的主要缺点。但是，卫星通信允许实现其特有的"多址连接"，因为通过一个通信卫星可以构筑以多个地球站为结点的网状的通信网(这多个所设置的器也称为无线中继器)。

7.3.3 网络接入技术

面向家庭、小型商务或小规模应用的主要接入技术有：

(1)普通电话公用网的接入：也称为 POTS(普通老电话服务)，比如一台 PC 机通过 modem 拨号经过公用电话网上网。采用的传输媒体是双绞线(电话筒线)，最大的传输速率为 14.4kbps、33.6kbps 以及 56kbps，接入质量较差。

(2)ISDN 接入：即窄带的综合业务数字网服务，其目标是希望使一个用户终端至另一个用户终端之间的传输全部数字化，以数字形式统一处理各类业务。目前主要通过 ISDNmodem(或者指定的其他相应设备)，利用拨号方式通过电话网，以 BRI 基本速率接口工作。采用的传输媒体是双绞线，最大传输速率为 64kbps 与 128kbps。一条线有可能连多台终端并提供多种业务(如话音、FAX、数据)

(3)ADSL 接入：ADSL 即是非同步数字用户环路的缩写，通过两种线路编码调制技术(无载波调幅调相和离散多音)，能在传统的电话线(双绞铜线)提供下行(至用户方向)6.1Mbps 的传输效率。事实上，XDSL 指的是不同种类的 DSL 技术，数字用户环路(DSL)技术是可通过普通电话线向用户提供高带宽信息传输的一大类用户接入技术。

(4)cable modem 接入：即采用电缆调制解调器在有线电视电缆上把数据进行调制,然后在线网的某个频率范围内传输，接收方在同一频带对信号解调整后获得数据。即采用有线电视网同轴电缆，其传输速率约为 2Mbps(512kbps 至 52Mbps)。

(5)低轨道卫星网接入：这是目前主要的无线接入技术之一。采用电视广播方式传输信息，传输速率可在 400kbps 以上。采用低轨道卫星可以显著减少地面与卫星之间往返传输的延时，但需要大批卫星才能覆盖全球，如计划中的 Teledesic 系统，需要由分布在 12 个平面上的 288 颗卫星组成。该系统计划下行速率为 64Mbps，上行速率为 2Mbps。

后面三种技术相对较新，有必要进一步说明。可以发现广大用户获取信息时的一个重要特征是：按收信息量(下行)远远大于发送(上行)量。比如因特网上 WWW 与 BBS 等最常用的服务，接收与发送量之比大于 10：1，FTP 服务更高达 100：1 以上。采用这类上行与下行不对称数据量传输的特性，可开发研制出性能价格比更好的新型接入设备，比如：

(1)ADSL 也称为不对称的数字用户环路，可以传统电话线路上提供下行至用户高达 6.1Mbps 的速率，特别适合于数据的高速下载与网上电视等业务的开展。上行 640kbps 也可大体满足用户发送请求和上递数据的需要。人们普遍认为 DSL 将是 21 世纪主流接入技术之一。

(2)cable modem 在物理层的传输机制与电话线上的 modem 大体相同，即采用调频或调幅对数据进行编码。可充分利用有线电视网(cableTV)四通八达连向千家万户共享介质，线路质量好，多频率带宽的优势。目前正在发展双向传输方式。在一些家用系统中也可以主要用于单向下行(达 2Mbps 以上)，上行使用电话线作补充手段。

(3)低轨道卫星接入也可以采用价格便宜并且实现方便的单向卫星接收系统，单向的下行接收速率可比标准的 28.8kbps 高出几十倍以上，如个人用户可获得 400kbps，小型公司用户甚至可达 2Mbps 带宽下载接收广播级质量的全屏幕视频文件。这时需补充采用地面拨号系统完 成上行任务。因特网卫星接入系统通常由最终用户方、服务提供商(ISP)和卫星系统三部分组成。最终用户只要配备卫星天线(如直径 0.6cm 左右)和计算机内的卫星接收卡(如 32 位 PCI 卡)或者加顶端盒组成一个接收系统，用于接收下载来自卫星的数据。采用地面拨号向 ISP 发出上行数据请求，ISP 把数据最终发送至卫星系统。

面向大公司、大型单位与机构至 ISP(指通信子网)的接入技术主要有：
(1)X.25 公用分组交换网接入：常采用租用专线方式，使用双绞线(如电话专线)，最高速率可达 64kbps。

(2)帧中继接入 ：采用来租用专线方式(双绞线或同轴电缆)，在功能上已可以替代 X.25，是一种高速流水线方式的分组交换技术，传输速率从 56kbps 可达 1.544Mbps 比 X.25 高得多。帧中继只提供了物理层和数据链路层的标准，更高层工作交给终端去完成。数据链路采用比 X.25 更简化的 LAPD(来自于 ISDN)的核心功能。

(3)光纤接入：光纤接入技术可分为光纤环路技术(FITL)和光纤同轴混合技术(HFC)。其中光纤环路技术采用全数字传输，依据光网络单元具体设置情况又可细人发为光纤到路边(FTTC)、光纤到大楼(FTTH)和光纤到户(FTTH)等。光纤接入可用于各类高带宽、高质量的应用环境中，比如以 DDN 专线和将来 B-ISDN 与 ATM 服务专线应用于大客户的接入。目前，我国 DDN 专线采用 2Mbps 为主。

7.3.4 规划与构建 WAN

在规划一个 WAN 时，必须着重考虑以下一些问题：

(1)选择相应的通信服务。因为使用通信子网和租用专线是一项代价很大的经常性支出，必须认真估算所需要的带宽。

(2)选择相应的接入设备与网络互联设备，其前期投入少。

(3)进一步做好 LAN 的构建工作。特别是大型企业集团构建性能良好的 LAN 是实施 WAN 的基本前提。

(4)选择 WAN 的构建方式，使处于不同地点的用户和 LAN 能有效地通信，保证必要的通信性能和质量。

(5)认真考虑 WAN 的安全性措施(见 7.6 节)。

(6)落实 LAN 与 WAN 的相应网络管理办法(见 7.7 节)。

1.两类重要的网络互联设备

在上一小节我们已经看到企业或事业单位接入通信子网时主要采用各种类型的 modem 和通信服务部门推荐的各类专用接入设备，接入设备首先连接的是路由器或通信服务器。

(1)路由器：路由器(router)是实现网络层互连的设备，在 OSI 标准中称为中间系统。这是一种应用相当广泛的提供网络服务设备。通常路由器提供了各种速率的多种链路或子网的接口，这是一种价格较为昂贵的主动的智能网络节点，一般参与网络管理，提供对资源的动态控制，支持网络工程的实现和协助网络维护活动。

①多于两个网络的网络之间的互联设备(图7.11左部显示了连接四个LAN的路由器)。目前已大多改变由交换机担任此功能，交换可快得多。

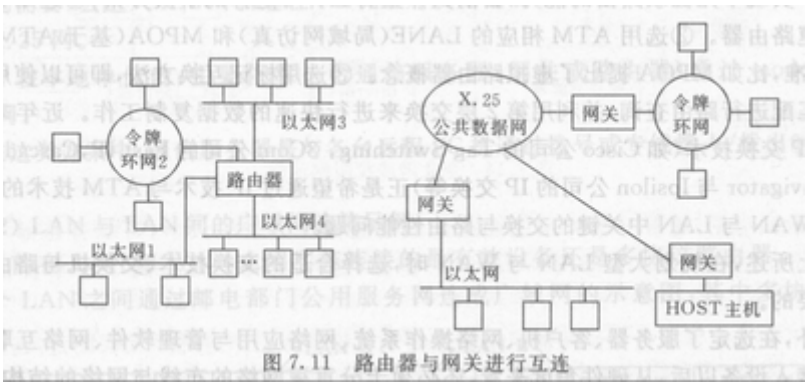


图 7.11 路由器与网关进行互连

②常用的WAN连接设备，包括允许采用各类WAN接口的多种路由器设备(见图7.13和图7.14)。

③在网络间的数据处理设备，其数据处理功能包括过滤、转发、优先权指定、多路复用、信息加密、数据压缩等。

④网络的一类管理设备，如包括网络中的配置管理、性能管理与容错管理。 用于拨号接入的路由器常被称为"访问服务器"。

(2)网关：网关(gateway)也称为信关，实质上是通信服务器。这是一种相当复杂的在应用层进行网络互连设备,可以用来连接的设备,可以用来连接异种网络(包括不同的各类网络操作系统),实现网络之间协议转换的功能(故有时也专门称为协议转换器)。

图7.11的右部显示了网关可以用于使两个或多个不同的网络之间相互连接进行通信，这里"不同"的含义是指可以允许传输协议与物理网络都不一样。

当我们限定使用几种固定协议时，也可以避开协议的复杂转换，而用路由器来替代网关(通信服务器)，承担起许多相应的WAN连接任务。

2.进一步规划大型LAN的构建

随着因特网的深入应用，用户对带宽的需求越来越迫切，大量业务跨越了多个网域，甚至已开始需要千兆位级的IP路由能力。

作为企业LAN和WAN中重要互联设备的路由器,目前其性能受到了两方面的制约：

(1)检查进入的数据报的目的地址，以此查询路由表决定下一跳。

(2)把进入的数据复制到相应的输出接口。
查询路由表意味着在一个很大的地址空间找出最长的前缀匹配。为了加速查询，通常的一个措施是采用交换式背板替代共享总线。

为了实现千兆位级路由的能力，目前大体上有三种可选择的解决方案：①选用相当昂贵的高速路由器。②选用 ATM 相应的 LANE(局域网访真)和 MPOA(基于 ATM 的多协议)标准，比如 MPOA 提出了虚拟路由器概念。③选用标记互换方法，即可以使用简单的严格匹配进行路由查询，并利用第 2 层交换来进行快速的数据复制工作。近年来正在兴起的 IP 交换技术(如 Cisco 公司的 Tag Switching,3Com 公司的 Fast IP ,Cascade 公司的 IP Navigator 与 Ipsilon 公司的 IP 交换等)正是希望通过 IP 技术与 ATM 技术的结合，来解决 WAN 与 LAN 中关键的交换与路由路由性能问题。

综上所述，在规划大型 LAN 与 WAN 时，选择合适的交换技术、交换机与路由器是特别重要的。

此外，在选定了服务器、客户机、网络操作系统、网络应用与管理软件、网络互联设备和网络接入设备以后，从硬件角度来看，还必须十分重视网络的布线与网络的结构问题。综合布线技术在 LAN 和现代化大楼中也有着突出的重要地位。一个完整的结构化结合布线系统大体上由垂直主干布线、水平支干线布线、信息终端、设备布线、管理枢纽、楼外干线等六个子系统所组成。标准化与规范化的综合布线可以提供话音、数据、图像、控制信号、视频等各类信息的传输，提供规范化的各类信息转接与接插方式，也可以为 WAN 的连接提供良好的基础。

3.WAN 的几种典型连接方式

分散的个别用户连入公共电话网可以采用 PC+modem 的拨号方式，这里将进一步以图例方式说明几种典型的 WAN 连接方式：PC 机远程存取本单位 LAN、LAN 与 LAN 间的广域网连接，以及企业网或校园网示例。

(1)PC 机远程存取本单位 LAN

图 7.12 是处于远程 PC 机通过 X.25 网络连接本地单位中的 LAN 示意图。

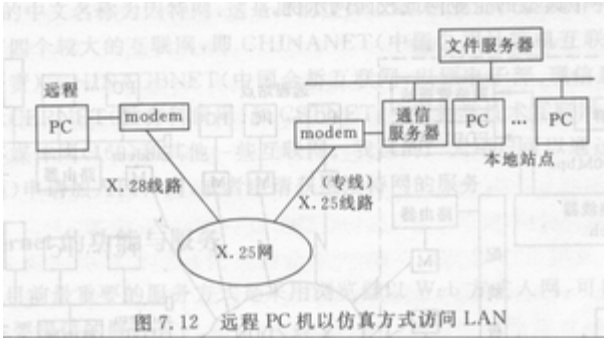


图 7.12 远程 PC 机以仿真方式访问 LAN

①在远程 PC 机上就装有相应的仿真通信软件，根据 X.28 规程的约定(因分组拆装 PAD 的需要)去进入 X.25 网；远程 PC 机最好使用电话专线(也可用电话拨号方式)申请连入 X.25 网。

②在本地单位内，主要依靠一台通信服务器(网关或路由器)通过 modem 接入 X.25 网。

在这类方案中，通信服务器与各台远程 PC 机通过拨号或专线拨入 / 拨出时的关键设备。

(2)LAN 与 LAN 间的广域网连接示例

目前在 LAN 与 WAN 之间实施连接的最有效设备还是多协议路由器。图 7.13 是两个 LAN 之间通过邮电部门公用服务网连成广域网的示意图，其中多协议路由器充当了关键的网际互联设备。通常，多协议路由器可以选购信誉良好的现成路由器产品。

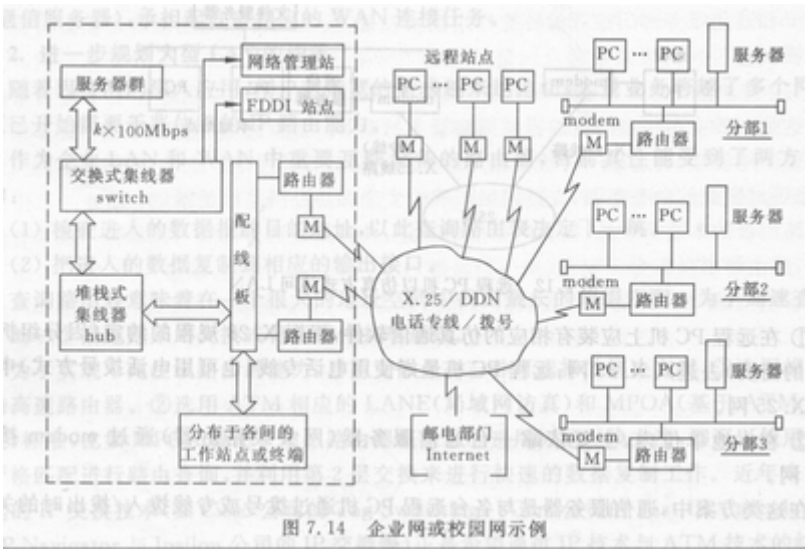


图 7.14 企业网或校园网示例

(3)企业网与校园网示例

图 7.14 是一个典型的企业网或校园示例。

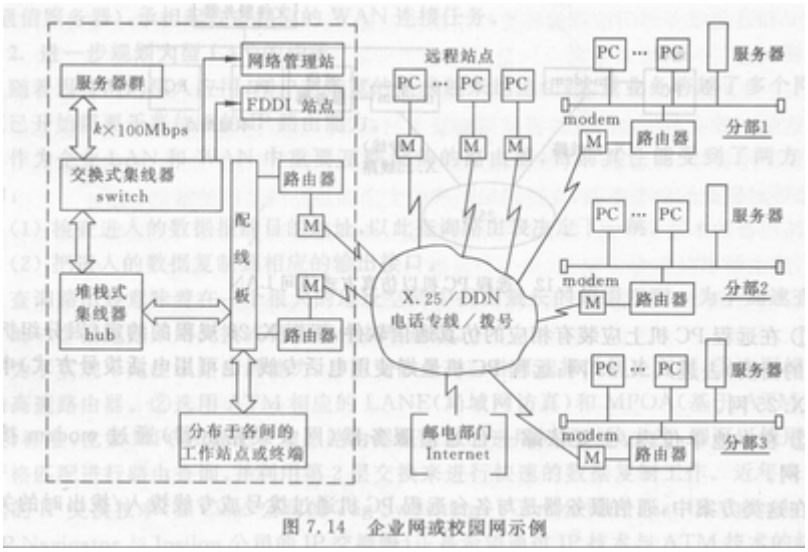


图 7.14 企业网或校园网示例

①由图可见，通常一个大型企业或地理位置相当分散的大学，可以有总部与若干个分部，还需要接通邮电部门的 Internet 服务，也可能有一批从属于本单位的远程站点。

②不论是总部或是分部均需采用路由器或通信服务器作为网际互联设备，路由器除承担网际互联作用外，还可起保护 LAN 网络安全"防火墙"作用，控制信息分组的出 / 入、检测传输出错。也可用于选择传输的路由。

③在 LAN 规模较大或带宽不足的场合下，必须使用交换式以太网技术，改进性能和保证带宽。

④在图 7.14 的左部，企业总部也采用了配线板方式的结构化布线系统思想来构成 LAN。其中"堆栈式集线器"是把每一 hub 作为一个模块组合起来使之方便管理与布线。有了配线板后 ，每一房间内的站点或终端可以通过预先布线安排好的"墙座"，十分方便地连在配线板上，通过接通 hub 端口或者选用路由器进行远程访问。

7.4 Internet 和 Intranet 初步

Internet 中文名称为因特网，这是目前世界上最大的计算机互联网络。

我国现有四个较大的互联网，即 CHINANET(中国公用计算机互联网，由邮电部门有关的公司负责)，CHINAGBNET(中国金桥互联网，以原电子部、现信息部下属公司为主负责)，CERNET(教育科研网)和 CSTNET(中国科学技术互联网)。此外 1997 年开始发展了多媒体网(169)和其他一些互联网。我国的广大用户可以通过接入 ISP(因特网服务提供商)申请加入因特网，或者申请获得因特网的服务。

7.4.1 Internet 的功能与服务

Internet 目前最重要的服务方式是采用浏览器 WEB 方式入网，可以获得大部分的服务项目。主要提供的服务有：

(1)远程登录(telnet),最基本的应用协议是 TELNET。

(2)文件传输(ftp)基本的应用层协议是 FTP

(3)电子邮件(E-mail)，基本的应用层协议是 SMTP(简单邮件传输协议)。

(4)网络新闻(news)，基本的应用层协议是 NNTP(网络新闻传输协议)

(5)查找文件(archie)。

(6)查找人员

(7)以菜单方式浏览信息(Gopher)。

浏览信息与查找不同。查找时用户一开始可能就有比较明显的目标，浏览在开始时可能是漫无目标的，也可能是目标相对模糊的，因而浏览服务通常都需要有引导或导航(navigation)的功能。Internet 中的 Gopher 软件就是通过菜单来引导的。另外，查找是由计算机程序来自动搜索的，而浏览则需要通过人和计算机交互，由人在程序的引导下不断搜索。用户在作出某些菜单项中可能包含自己所需信息的推断之后，必须再一一考虑这些选项，以确定其中包含有属于自己所需要的内容。

(8)按内容自动查找(WAIS)。

这里所做的查找和前面的"查找文件"中的查找不同之处在于，前者是按名自动搜索的，而这里却可以按内容来自动搜索，因而搜索的对象往往是已作关键词索引的数据库。Internet 中的 WAIS(wide area information server)就提供这样的功能。事实上，目前，在因特网上已发展了一批搜索引擎，提供相应的导航服务(如 Yahoo)。

(9)全球范围的超媒体信息浏览服务(WWW)。

目前广泛在因特网上使用高级浏览服务，把存入于全球范围内众多计算机上的信息以超媒体的方式链接在一起，构成一个范围的网，称为 WWW(world wide wed).WWW 之所以成功，在于它制定了一套标准化且易懂的超文本描述语言 HTML(hyper text makeup language)、超文本传输协议 HTTP(hyper text transmission protocol)URL(uniform resource locator).LRL 规定一个文档在 WWW 中存放地点的统一的资源定位格式，地址。例如：
http://www.cryst.bbk.ac.uk 就是一个 URL 的地址。其中第一部分 http 表明传送所遵循的协议，此处即为 HTTP，但是它也可以为 ftp 或 gopher 等；第二部分是信息资源所在的计算机域名，在它后面还可跟随目录名和文件名。不加目录名和文件名，不加目录名和文件名，通常就表示主页(home page)。在浏览 WWW 时

获得的是一个超媒体的文本，其中醒目词条或带有下划线的词条实际上就是一个链接，指向另一个 URL 地址，从而进一步访问其他的信息资源。通过超媒体链接，用户几乎可以访问到任何一种 Internet 资源。

早期最有名的用于访问 WWW 的软件是由美国超级计算机应用中心(National Center for Supercomputer Applications,NCSA)开发的 Mosaic。用户可以通过 Mosaic 借助鼠标来浏览全球范围的多媒体信息。目前更为流行、用户界面更好为友好的浏览器 (browser)是Netscape 开发的 Navigator 和 Microsoft 公司开发的 Internet Explorer.

7.4.2 Internet 的协议和 IP 地址

企业的 LAN 只要与 ISP 连上，并申请和注册好相应的域名(和 IP 地址)，就算加入了因特网大家庭。因特网采用的是 TCP/IP 体系结构网络协议，加入因特网的任何网络必须支持 TCP/IP，即在互联网层采用 IP 协议，在传输层采用 TCP(或 UDP)。所有因特网上的协议都采用 RFC 加上数字编号的方法命名和公布。

事实上因特网的应用层协议发展较快，前面提到过早期，TELNET,FTP,SMTP(电子邮件)等，进一步还提出了 DNS(Domain Name service)域名服务协议、NSP(Name Service Protocol)名字服务协议、HTTP(Hyper Text Transfer Protocol)超文本传输协议等一大批与因特网上应用服务相关的协议，此外还可以有网络管理(如 SNMP)和网络安全等方面的协议。

在遵循 TCP/IP 协议的前提下，任何网络和用户都要以方便地采用前面已讲过的 WAN 接入的任何方式去接入因特网。顺便指出，远程拔号上网时，早期 UNIX 系统采用 SLIP(Serial Link Internet Protocol)串行链接互联网协议，目前大多已采用具有寻址、分组标识和压缩功能的 PPP(Point to Point Protocol)点到点协议(RFC1661～1663)，就是一个数据链路层协议，允许连接不同的网络层协议。

在采用 TCP/IP 协议互连因特网时的一个基本概念是 IP 地址和域名。连接到 Internet 上的每台计算机都必须有一个惟一地址。Internet 的地址是用数字来表示的，称为 IP 地址。IP 地址含有四个字节，32 个二进制位。在书写时，通常每个字都用十进制表示，而字节之间用小黑点"."分隔开来，例如，IP 地址可以是 128.5.1.0 或 35.1.7.48。IP 地址又分为五类：A 类、B 类、C 类、D 类和 E 类。大量使用的仅为 A，B，C 三类。A 类地址中第一字节表示地址，而后三个字节表示该网内计算机的地址；B 类地址中前两个字节表示网络地址，后两个字节表示网内计算机的地址；C 类地址则是前三个字节表示网络地址，后一个字节表示网内计算机的地址。A 类地址用于非常巨大的计算机网络，B 类地址次之，C 类地址用于小网络。目前已经在考虑新的 IP 协议，包括新的编直址方法(如 IPV6 协议)。

接入 Internet 的某台计算机要和另一台计算机通信就必须确切地知道其 IP 地址。要记住这么多枯燥的数字不是容易的事情，人们更习惯于使用字母表示的名字。Internet 上的每台计算机都必须具有惟一的名字，才能区分开来；因此，

Internet 上计算机的名字通常由许多域构成，域间用小黑点"."分隔。例如，mars.ee.fudan.edu cn。这种名字有时又称为域名(domain name)。对域名中的倒数二个域有些约定，可以区分机构或组织的性质。常用的有：

edu 教育机构

com 商业机构

mil 军事部门

gov 政府机关

org 其他组织

在美国，通常就用他们作为最后一个域。而在美国以外的其他国家和地区则用标准化的两个字母的代码来作为最后一个域，如 cn 表示中国，hk 表示中国的香港地区等。

名字仅是为帮助人们记忆和输入用的，在 IP 分组中使用的仍然是 IP 地址。Internet 中的许多称为域名服务器(Domain Name Server,DNS)的系统要可以帮助用户自动地从域名来找到其的地址。当某个本地的域名服务器找不到相应的地址时还会与其他的域名服务器联系来进行查找，而且这个过程对用户是透明的。

通常采用 TCP/IP 协议中的 ARP 协议和 RARP 协议进一步进行 IP 地址与主机(网卡)物理地址的转换。

7.4.3 Intranet 初步

随步 Internet 的广泛使用与显著进步，企业网络从中获取了经验与技术，促进了企业内部网 Intranet 的迅速发。可以说，现代的企业网络将以 Intranet 作为基础。

Intranet 的中文名称为内联网，尽管网络专家们对 Intranet 下过多种稍有不同的定义,但是人们一致认为 Intranet 是把 Internet 技术用于企业局域网的产物，也是汲取了 Interne 和企业局域网两者的长处而有机地组成的新型企业网络。

一个企业的 Intranet 网可以实现可以下的一些主要功能：

(1)允许企业的 Intranet 根据企业自身的意图，按需要接通或断开与 Internet 的连接：可以根据自身的利益，决定如何去充分利用 Internet 上的信息资源和各类服务，决定如何通过 Internet 有控制地或有限制地公布或分发企业信息，比如通过 E-mail、news 主页。

(2)允许企业采用 Web 技术，实现 browser/serve 模式的工作方式。通过浏览器和 HTTP 协议，向企业中的各类用户提供相对统一的用户界面。使用这类方便的

界面，用户可以获得各类信息服务，直接或间接地访问各类服务器。换句话说，浏览器所提供的友善的使用工具，有可能成为企业内部所有的 LAN、有关的 WAN 和 Internet 的强有力的手段。

(3)在企业的 Intranet 内，可以有效地向用户提供下列服务：数据库服务、文件服务、目录服务、电子邮件服务、命名服务、引导服务、打印服务、对象服务、浏览服务、网络管理服务、安全服务。

(4)允许企业采用 TCP/IP 协议组为主要的网络连接协议，接通整个企业范围内的所有必要的工作站点和全部相应的服务器设施，实现一个功能相对完善的企业网络。

(5)尤其是针对 Internet 中令人感到不安的信息安全与网络管理的很多问题，企业 Intranet 可以实现相对良好的网络安全保障措施和相对健全的网络管理功能.

概括起来说，Internet 技术的主要特征表现在：①采用 TCP/IP 通信协议作为企业网络通信的基础，②采用 Web 技术和 HTTP、SMTP、FTP、SNMP、DNS 等一系列公开协议标准作为企业网络构建时的基本技术，③主要着眼于满足企业内部的使用，在此基础上根据领导层的决策有控制地向外发布主页信号以及向企业外部提供有限的信息服务。④在企业 LAN 的基础上增扩了 WebServer、Mail Server 、FTP Server 和 DNS Server 等主要软硬件设施，构成完整的 Intranet 网络。⑤采用了有效的网络安全设施和网络管理平台，保证 Intranet 的运行效率和企业的根本利益。

7.4.4 Extranet 概念

大约在 1996 年下半年出现了 Extranet 这个新名词。Extranet 的中文名称为外联网，是企业外部网的意思，是 Intranet 概念的进一步延伸与扩展，即把 Intranet 构建的技术应用到了企业与企业之间的网络互联上，成为企业之间合作的纽带或桥梁。

更具体地看，Extranet 把采用 Web 技术构建的企业信息系统的适应范围进一步扩大到了一些特定的外部企业，比如允许把本企业有关的主要公司、商店、常规固定客户、业务交易场所等也纳入到信息系统之中。

构建 Extranet 所采用的技术大体地雷同于 Intraner，即以 Web 技术、通信互联技术和数据库技术为核心，但是需要进一步保证实现下列目标：

(1)使外部企业与外部用户能随时按需要连接上本企业的 Intranet。

(2)保证与外部企业通信传输过程中的安全性，信息在网络途中需要采取必要的保护措施并保证到达目的地。

(3)保证有一定的响应时间性能，能满足实际业务的传送时限要求。所选用的网络服务应在通信费用的约束下能保证响应功能。

(4)防止或阻断来自外部的非法访问。比如可以使用双防火墙技术，在两个防火墙之间的网络称为"非军事区"，Extranet 对外部企业所用的服务器可设在非军事区内。

(5)对访问的内容作相应的权限控制。即根据不同的用户对象，分别给予不同的服务与可查看信息的范围，并能允许动态地加以改变。

这些都是构建 Extranet 时应考虑的关键技术。

目前主要有三种类型的 Extranet：第一类是基于数据库服务为主的，第二类是基于联机事务处理服务为主的(OLTP)，第三类则是基于群件应用为主的(即多个企业协作完成必要的工作任务流)。也许将会出现或流行"虚拟企业活动"方式的 Extranet。

从上面的叙述可见，Extranet 的安全策略是一个十分重要的方面。目前大体上有可取的三各模式：

(1)通过企业与 ISP 的密切合作来建立与管理 Extranet。即企业委托专业 ISP(因特网服务供应公司)或专业大型计算机网络公司为它设计并管理 Extranet，则 ISP 全权负责通信子网的有关接入与管理工作，保证 Extranet 的安全性。

(2)企业委托专业加密与认证服务中心负责安全事宜，用户信息经中心鉴定与解密后，由专线进入本企业的 Intranet。在企业内部则进一步根据不同用户限定的访问控制。

(3)全部由企业采用网络安全协议新技术(如 SSL/SHTTP)，结合专线 VPN(虚拟专用网)与防火墙技术等各类措施自行构建 Extranet。在构建过程中由 ISP 和网络专业司给予必要的指导和帮助。

发展网络服务将是进一步推进 Extranet 的重要前提。比如因特网上的 VPN 服务、支持 TCP/IP 协议的保密帧中继服务、专门为商业服务提供商业专用网络的"开放型商业网"服务等。都可能是有效的 Extranet 所需要的通信子网服务形式。

7.5 客户机/服务器模式与网络计算

本节将进一步讨论网络应用的各种实用方式，包括文件服务器模型与域模型、客户机/服务器模式、网络计算与电子商务。

7.5.1 文件服务器模型与域模型

目前 LAN 中仍普遍使用文件服务器、工作组和域模型等方式。

1.文件服务器(或打印服务器)为主体的使用方式

美国 NOVELL 公司的 NETWARE 局域网(操作系统)是取得极大成功的典型代表。在这类 LAN 中，以一台(或多台)文件服务器作为共享资源的主体，在文件服务器中提供了可高速存取的大容量磁盘，其中驻留着所有网络中的工作站点(如 PC 机)可以共享文件与目录。LAN 中的每个用户可以在自己的工作站点上，根据允许的权限以"文件"为单位去存取(读/写)在文件服务器硬盘中的内容，包括程序文件、数据文件、数据库文件和有关的网络信息文件。至于在 LAN 上有哪些是合法用户，哪个用户有何种权限可以去读出、写入、修改、删除等有所有有关信息，也都驻留在文件服务器的有关"机密"信息库内，只有授权为网络管理员或超级用户的特定管理网络人员才可以去访问与建立这些"机密"信息库。

必须指出的是，如果在某个应用任务中，LAN 网络上有多个不同的用户或不同作业要去同时改写或修改在文件服务器内的同一个共享文件时(如数据库应用中经常会遇到这类问题)，那么只能依靠在应用程序中严格规定的"纪律"或"约定"才能妥善地加以处理，即对应用程序本身提出了严格的编制要求。

2.以工作组为主体的使用方式

在这类 LAN 使用方式中，由某个拥有特权的系统管理员创建一个用户工作组，其中包含了 LAN 的用户要使用的大部分资源。原先的工作组模型，通常是指分布式的点到点(也称为 peer to peer，端到端对等方式)的网络，即网络中的资源与管理均可以灵活分布在 LAN 之中，工作组中的每一台计算机都可以用作为服务器和工作站，并且具有自己的帐户与管理。

工作组模型的优点是：资源可以相当随机与灵活地分布，更方便于资源共享，管理员只需去实施相当低级的维护；设计与实现该类 LAN 十分方便，能使紧密相邻的 2-6 台有限数目的工作站(网上邻居)十分方便地互连。其缺点也十分明显：缺乏集中管理与控制的机制，没有集中的统一帐户管理，未能对资源实施更加高效率的集中管理，也未能实施工作站的有效配置与安全性严密控制。不少工作站可能联网效率很低，还需要监控每一台计算机上的帐户，在整个 LAN 上存在着账户的重复，增删用户也十分不便，因此只适合于小规模用户的使用。

3. 以域模型为主体的使用方式

这是 Microsoft 公司的 Windows NT Server（或 Windows 2000 Server）和 IBM 公司 OS / 2 LAN Server（或 Warp Server）等系统中着重推行的使用方式。事实上在一个较大的公司中可以有几个部门，其中每一个部门中所有的计算机连接成为一个域（即整个公司可划分成几个域），每一个域中的所有用户又可细分为若干个用户工作组。可以认为域模型是相对更先进更有效的 LAN 方案，其中包含了“集中的管理与集中的安全控制”，即有效地汲取了文件服务器的优点。

在一个域中，可以包含有联网的一批服务器和工作站点，其中至少有一台是域服务器（也称为域控制器）。域服务器承担了域内集中式的管理与安全职能，即有一个由系统管理员控制与创建的安全账户管理数据库，在这个域环境中将一批计

算机作为逻辑上的对象组合在一起（如该部门中的所有计算机及设备），称为一个域。

采用域模型时，也提供了定义用户环境的有效手段，即允许管理员为每一个用户（或用户组）创建一个用户登录文件（profile），从而可确切地规定每个 IAN 中的用户具体登录时的工作环境与使用规则。既可以帮助不熟练的用户避免发生差错，也可以规范整个 LAN 的使用环境。进一步看，为了整个域的安全可靠工作，还可以在域内设置备份域服 务器作为主域服务器的动态备份，一旦主域服务器发生重大意外，整个域还可继续正常工作。

7．5．2 客户机／服务器模式

20 世纪 90 年代以来，客户机／服务器（client／server）模式（以下简称为 C／S 模式）得到了十分迅速的流行。几乎每个新的网络操作系统和每个新的多用户数据库系统都声称能支持 C/S 用模式。实现 C／S 模式允许有许多不同的策略。

从数据库管理系统的应用来看，在 LAN 上采用的 C／S 模式，即指在 LAN 中至少有一台数据库服务器（DBMS server），可以作为希望去存取公共数据库的各台工作站的后援支持。把应用任务中的程序执行内容划分成两部分：与数据库存取有关的部分由 DBMS server 承担，与应用的人机界面处理、输人／输出或一部分应用的逻辑功能等有关的内容由 client 端工作站承担。这样做的好处大致上有：①充分调动在 LAN 中的 server 与 client 两方面的处理能力；②极大地减少网络上的信息流通量（可不再以整个“文件”为传送单位，采用请求一服务响应的方式，网上仅传输经 server 加工处理后的那一部分必要的结果信息）；③有效地发挥了服务器软硬件执行效率高、集中管理数据库安全方便的长处，也可以充分利用 PC 机 client 端处理用户界面（特别是图形用户界面)和本地 I/O 的优点；④C/S 体系结构有可能提供一种开放式的、易于伸缩扩展的分布式计算环境，并保护硬件等投资。

近年来已普遍采用了三层方式的 C/S 模式，即客户机——功能服务器或应用服务----数据库服务器。把应用系统的软件相应的分为三层:①客户机实体内驻留用户界面层(也称为表示层）软件，负责用户与应用之间进行对话的任务。②功能服务器实体内存放有业务逻辑层（也称为功能层）软件，用来响应客户机的请求，完成相应的业务处理或复杂计算任务.如果有数据库访问任务时,则可进一步向数据库服务器发送相应的 SQL 语句。③数据库服务器实体内驻留有数据库服务层（也称为数据层）软件．用来执行功能层发送来的 SQL 语句，负责管理对数据库数据的读写、数据库查询与更新等任务。任务完成后逐层地返回给客户机上的用户。

采用三层 C/S 用模式的好处是：①可以更方便、更清晰地分工应用软件的设计任务。②可以降低对客户机的要求，使客户机只需处理人机界面为主的工作，适应日益扩展的应用需求。③防止客户机上有权连接数据库的用户绕过系统中的客户端应用系统，利用自行安装在客户机上的数据库访问工具非法访问某些未授权的数据，从而保证了安全性油功能服务器把关）。④避免了在客户机上分发应用程序与版本控制上的困难。

7．5．3 网络计算初步

通常认为，网络计算是一种高效率和低成本的计算模型，这种模型适合于充分地利用因特网的功能，提供了一类一体化的解决方案。

也有不少专家与用户认为 Intranet 的构建，已为在企业中实现网络计算提供了基本的条件，在 Intranet 环境下网络计算将以 Web 技术为基础，采用 browser / server 结构，结合数据库技术，形成跨平台开放性的具有多媒体应用特征的信息服务和数据库服务。

普遍认为，网络计算中所依托的主要基本技术，除了原有的 TCP / IP 网络协议体系结构与数据库技术外，目前已有：
（1）Web 技术和 HTTP 协议标准（包括 CGI 公共同关接口和进一步的各类技术）；

（2）HTML（超文本描述语言）及有关的主页制作技术（包括进一步的发展 XML 等）；

（3）瘦客户机技术；

（4）Java 语言编程技术和应用技术，也可包括 Java ScriPt；

（5）browser / server 结构模式。

在实际使用时，基于 B / S 模式的信息系统通常会采用三层或更多层的结构。浏览器——Web 服务器——数据库服务器。以 Web 服务器作为系统的核心，用户端通过 browser 向 Web 服务器提出查询请求（HTTP 协议方式），Web 服务器根据需要再向数据库服务器发出数据请求。数据库服务器根据检索与查询条件返回相应的数据结果给 Web 服务器，最终 Web 把结果翻译成 HTML 或各类 Scripts 语言的相应格式发回至 browser，用户通过 browser 浏览所需结果。

目前几乎各大数据库厂商都已开始提供了具有 B / S 模式工作能力的数据库新产品。事实上，从 Web 访问数据库时可以使用多种技术，包括：CGI 技术（在 Web 服务器上运行具有某种功能的 CGI 程序）、Web API（应用编程接口）技术（如微软公司的 ISAPI，Netscape 公司的 WSAPI 等）以及 JAVA 语言访问数据库的接口（如 Sun 公司的 JDBC）。

进一步看，使 Web 与数据库相结合的 Web API 技术功能强，性能好，使用灵活，但是对开发人员要求高。相应的软件公司进一步在各自的服务器中提供基于 Web API 技术的更方便使用数据库的接口，如微软公司在 NT 的 IIS（因特网信息服务器）上提供了一个动态链接库 IDC（因特网数据库连接），后来又进一步提供了更好的 ASP（active server Pages，主动服务器页面）技术等，允许采用 ADO（ActiveX Data Object）通过 ODBC 访问各种数据库。

7．5．4 电子商务介绍

电子商务（electronic－Business 或者 electronic commerce，eB）通常指的是商务贸易活动中各个环节的电子化，其内容可覆盖与商务活动有关的所有方面。更具体地说，eB 是指通过计算机网络进行的生产、营销、销售和流通活动，不仅指在因特网上的交易，而且也包括利用信息技术门）来降低商务成本、增加流通价值和创造商业机遇的所有商务活动。比如通过网络可能从事于各类产品与原材料的查询、采购、产品展示、订购直到出货、储运以及电子支付等一系列贸易活动。eB 是一种典型的网络计算模式的应用。

随着 Internet 和 Intranet 的迅速发展，eB 活动已开始基于因特网的广泛使用而崛起，它大体上可分为三大类：

（1）政府贸易管理的电子化：在 EDI 基础上有了质的飞跃与提高，即采用基于因特网的网络技术实现数据与信息的处理、传输与存储，除了取代传统的纸面单据外，提供更为广泛的信息服务与消息发布等功能。比如海关、外贸与内贸管理机构、商委等都在考虑采用 eB 的方式工作。

（2）企业与企业之间的电子商务：可基于 Internet，Intranet 或 Extranet，采用网络技术实现与供货商、生产厂商或用户之间的有关商务活动，比如下订单、电子支付、展示产品目录、储运交接以及检查合同执行情况等。一般可视为 B TO B 方式。

（3）企业与个人消费者之间的交易：即电子购物，企业通过网络为个人消费者提供服务和实施相应的商业行为，比如可方便地展示商品的用途、质量、价格、定货方式、付款办法等，实现在线购物。常称之为 B TO C 方式。

事实上，eB 活动的开展受到了多方面的制约，比如：电信资费的开支、信用支付体系的保障和上网用户的数量等，即使企业已建成良好的企业内部网 Intranet，开展 eB 仍需许多外部条件的配合。

从技术角度来看，eB 将以 Internet、Intranet 和 Extranet 建设为基础，采用网络计算模式开展商务活动，因此网络传输与信息的安全性技术有着关键性的重要作用。

比如：世界上最大的计算机公司 IBM 公司正在积极投人电子商务活动的推广工作。IBM 公司基于因特网的电子商务技术主要推出了 Websphere 电子商务平台等。

7.6 网络的安全性

7．6．互联网络安全性的基本概念

对于一个计算机网络来说，网络的安全性增加了信息安全与路由安全等方面的许多问题。比如，在网络环境下可能危及信息安全的因素包括：

（1）伪装欺骗：指非法用户假冒合法用户身份获取敏感信息的行为；

（2）非法存取：窃取、篡改或破坏网络中存储的信息；

（3）否认抵赖：指通信方事后否认曾参与某次活动的行为；

（4）服务拒绝：指合法用户的正当申请被拒绝、更改或延迟等；

（5）截取破译：指以不正当手段获取口令或密钥，或者对加密信息与加密机制进行分析破译或篡改等。

危及路由安全方面的因素可能更多一些：

（1）侦听窃密：在通信信道中窃听传输的信息；

（2）指定路由：非法用户精心设计一些绕过安全机制的路由或错误引导信息送向不正常的路由；

（3）身份攻击：指用户的身份或口令在通信时被截取；

（4）截取破坏：非法用户截取通信过程中的数据，或者伪造、篡改与破坏信道中传输的信息；

（5）中继攻击：非法用户截取信息后延迟时间再发送；

（6）阻塞通信：利用网上的软硬件漏洞，发送大量无效信息使网络过载，或者设法使一些路径或网上关键互联设备瘫痪。

ISO 7498－2 网络安全体系确定了多种基本安全服务和多种安全机制，在 OSI 七层中分别实现网络的安全服务。概括起来说，网络的常用安全机制有：

（1）标识与验证机制：主要指对用户身份的标识与鉴别，比如用户名加上口令是最基本的方法。这也是网络访问控制的药剂。

（2）网络访问控制机制：主要指对具体的各个用户给予必要的授权，允许或限制其访问存取的范围。方式、时间、地点与可采用的操作。可以根据本单位的安全策略进行设置与控制。

（3）加密机制：确定对不同的信息类型采用和实施不同的加密措施，包括存放在存储媒体内的数据和需要进行传输的数据。

（4）信息完整性机制：指的是判断并确认数据或信息在存储或传输过程中是否被篡改或破坏。通常加密机制与信息完整性机制都是采用密码技术来实施的。

（5）认证和审计机制：确认接收到的信息的真伪，监视与记录对网络的访问，发现或预防网络安全性方面的漏洞。

7．6．2 网络的信息安全技术

1．访问控制与目录管理

资源的合法使用主要依靠网络管理员进行必要的访问控制与目录管理，用以保证网络资源不被非法用户使用或破坏，也保证数据资源不被非法读出或改写。

根据单位的安全性策略，网络管理员应当十分熟悉由网络操作系统、数据库管理系统或相应网络安全管理软件等提供的手段，进一步实施访问控制与目录管理方面的措施。

2．数据加密

随着密码技术的进步，已发展了许多数据加密算法与技术。最有名的是 IBM 公司提出的 DES（数据加密标准）加密算法，1977 年已成为美国标准。这是一种对称加密的算法，采用了一个 56 位密钥对 64 位的数据块进行加密。"对称"是指采用的保密密钥既用于加密，也用于解密。

比如一种三重加密 DES 算法的执行过程大体上是：把 64 位的明文通过@函数处理后，经过 16 步的加密步骤处理（采用 56 位的密钥，使用复杂的变位处理，形成每一步有不同的密钥），再把处理结果的左右 32 位进行对换，最后经@函数的逆变换获得 64 位密文。目前已可利用专用芯片来实现 DES 算法。

RSA 算法是比较有名的一类非对称加密算法，但由于算法运算时速度较慢，大多用于信息量较小的加密场合。所谓"非对称"是指把密钥分开为一对，一个公开密钥用于加密，另一个专用密钥用于解密，便于密钥的管理。

目前在 Internet 上提供了另一种 PGP（Pretty good Privacy）加密系统，把加密技术交给了公众使用。PGP 发源于 E－mail 的加密，其中提供了私有保密性、身份鉴别、数字签名和压缩等各种技术措施。基本上采用现有的一些加密算法（如 RSA、IDEA、MD5）等加以综合而成。

网络传输过程中的数据完整性还可以用加密技术加上消息摘要技术来实施。比如采用"单向散列函数"计算出信息的"摘要"，连同加密后的信息一起发送给接收方；接收方则重新计算"摘要"，并进行比较，可以验证信息在传输过程中的完整性。最常用的数据摘要算法有采用 DES 算法实现的 X9．9 算法和由 RSA 公司设计的 MDS 算法（网上可免费提供）。

3．身份验证与鉴别

不仅对用户身份进行验证与鉴别，也可以对信息的真实可靠性进行验证与鉴别，用来解决冒充、抵赖、伪造或篡改等问题。除了标识用户加上口令的机制外，最常用的是"数字签名"技术。

数字签名可以采用秘密密钥或公开密钥技术实现，也可以使用上段所述的"消息摘要"方式配合实现。签名算法可用 RSA、DSS 数字签名标准等。比如一种数字签名的实现方式可能是：发送方从报文文本生成一个 128 位的散列值（或者消息摘

要），再用专用密钥对这个散列值加密形成一个发送方的数字签名。把签名与报文一起发送给接收方时，接收方可以从接收到的原始报文中计算出 128 位的散列值（或者消息摘要），再采用发送方的公开密钥对报文的数字签名解密后进行比较，如果相同，即可确认该数字签名确实是发送方发来的。

考虑到身份验证与鉴别中可以采用相当复杂的算法或技术，目前已开始流行委托专业认证鉴别机构或公司进行身份鉴别与验证，目的是采用更先进的技术做好认证与鉴别这一关键性的安全保卫工作。

RADIUS（远程拨入用户认证服务）是一种管理标准，可为分布式网络上的认证服务器提供管理服务，用来实现集中管理方式下的用户认证、口令加密、服务选择、信息过滤以及账户核对等有关任务。一个单独的 RADIUS 数据库服务器可能在多个网络上同时管理多个安全系统，其中保存有与网内所有用户有关的认证信息（如存取限制、特定的路由表、数据分组过滤规则或约定、可供核对的账户信息等），可以用来维护成千上万个用户的信息安全。

4．两个 TCP / IP 安全协议

TCP / IP 协议的安全机制先天不足，如 IP 地址由软件配置，存在有假冒地址和欺骗地址的隐患；支持的路由方式也存在着路由攻击的隐患。此外，上层应用协议（如 FTP，TELNET，SMTP 等）也缺乏认证与保密措施。因此，随着因特网的广泛使用，有必要提供新的完整的安全协议。

Kerberos 是为分布式系统提供的认证方案，能为每种服务提供可信任的第三方认证服务。这是由美国 MIT（麻省理工学院）为雅典娜工程设计的安全方案。在 Kerberos 系统中只有事先在其中登记过的客户才可以申请服务。客户希望得到某一服务，需要向认证服务器申请一张能使用此服务的入场券（ticket），然后该客户把入场券传给入场券分配服务器（TGS），实施入场券的检查。在得到证实以后，才能使用相应的服务。其中加入了个人身份确认机制以及客户与服务器通信中的数据加密。通过系统维护数据库的方式，保存着客户、服务、权限与密钥等方面的相应信息。

SSL / SHTTP 技术是在电子商贸应用中为了保证网上安全性而发展起来的。
Terisa 公司开发的 SHTTP 是在 HTTP 协议基础上加以扩展而成的，其中还增加了安全功能，主要限于维护与 Web 服务器之间的通信。在 SHTTP 协议中允许以各种方式封装信息（加密、签名与认证），还定义了对头信息进行密钥传输、认证传输和相应的过滤功能，并且为程序员提供有效的编程环境。

网景公司（Netscape）设计的安全套接层（secure socket layer，SSL）协议是一类因特网通信标准，用来提供两个应用之间通信的保密性、可信性与身份认证。SSL 是位于传输层之上的协议，即在四 SSL 上可执行 TELNET，FTP，HTTP 等应用程序。采用公开密钥技术为主，提供三类基本的安全服务：信息加密服务（在客户机与服务器之间所有业务都使用握手协议、协商密钥与算法进行加密）、信息完整性服务（采用机密共享与 HASH 函数组确保 SSL 业务可正常全部到达目的

地）、相互认证服务（采用公开密钥编码标识符，使客户机与服务器在 SSL 握手时交换各自标识符实现互相认证）。

Kerberos 和 SSL / SHTTP 都是完整的安全协议，作为保证通信信息安全的协议应提供下列功能：双方确认认证、数据源的不可否认性、数据接收方的不可否认性、信息完整性保护、以密码保密信息内容以及防重传性（指一个信息包被获取后不能再被重新传输）。

7．6．3 防火墙技术

防火墙是一种运行专门的计算机安全软件（称为防火墙软件）的计算机系统，即通过软件与硬件相结合，能在企业内部网络与外部网络之间的交界处构造起一个"保护层"，所有的企业内部网络与外部网络之间的通信都心须经过此保护层进行检查与连接，只有授权允许的通信才能通过保护层。换句话说，防火墙相当于一个安全网关，能在一定意义下阻断或隔离企业内部网络与外部网络，防止非法的人侵行为或破坏行为。用防火墙可以阻止外界对内部网资源的非法访问，也可以防止内部对外部外的不安全访问。

实现防火墙的新产品主要两大类：一类是网络级防火墙，另一类是应用级防火墙。目前一种趋势是把这两种技术结合起来。

（1） 网络级防火墙：也称为包过滤型防火墙。事实上是一种具有特殊功能的路由器，采用报文动态过滤技术，能够动态地检查流过的 TCP/IP 报文或分组头，根据企业所定义的规则，决定禁止某些报文通过或者允许某些报文通过，允许通过的报文将按照路由表设定的路径进行信息转发。相应的防火墙软件工作在传输层与网络层。

（2） 应用级防火墙：也称为应用网关型防火墙，目前已大多采用代理服务机制，即采用一个网关来管理应用服务，在其上安装对应于每种服务的特殊代码（代理服务程序），在此网关上控制与监督各类应用层服务的网络连接。比如对外部用户（或内部用户）的 FTP，TELNET，SMTP 等服务请求，检查用户的真实身份、请求合法性和源与目的地 IP 地址等，从而由网关决定接受或拒绝该服务请示对于可接受的服务请示由代理服务机制连接内部网与外部网。代理服务程序的配置由企业网络管理员所控制。

（3） 目前常用的应用级防火墙大致上有三种类型，分别适合于不同规模的企业内部网：双穴机网关、屏蔽主机网关和屏蔽子网关。一个共同点是需要有一台主机（称之为堡垒主机）来负责通信登记、信息转发和控制服务提供等任务。

① 双穴主机（dual-homed）网关：由堡垒主机作为应用网关。其中装有两块网卡分别连接外因特网和受保护的内部网，该主机运行防火墙软件，具有两个 IP 地址，并且能隔离内部主机与外部主机之间的所有可能连接。

② 屏蔽主机（screened host）网关：也称甄别主机网关，在外部因特岗网与被保护的企业内部网之间插入了堡垒主机和路由器，通常是由 IP 分组过滤路由器

去过滤或甄别出可能的不安全连接，再把所有授权的应用服务连接转向应用网关的代理服务机制。

③ 屏蔽子网（screende subnet）网关：也称甄别子网网关，适合于较大规模的网络使用。即在外部因特网与被保护的企业内部网这间插入了一个独立的子网，比如在子网中有两个路由器和一台堡垒主机（其上运行防火墙软件作为应用网关），内部网与外部网的一方各有一个分组过滤路由器，可根据不同甄别规则接受或拒绝网络通信，子网中的堡垒主机（或其他可供共享的服务器资源）是外部网与内部网都可能访问的惟一系统。

防火墙为 intranet 提供了较好的安全性防护，便也带来了一些问题，比如：

(1) 堡垒主机工路由器本身的安全性至关重要，也可能成为影响网络性能的瓶颈。

(2) 防火墙在一定程序上限制了服务的类型和用户应用的灵活性，有时会阻断一些人们常用的服务。

(3) 防火墙难以防止某些内部人员的攻击，也不能防止从后门进入内部网的非法访问（如通过拨号 SLIP/PPP 连接），对一些人们不熟悉的应用或软件，包括病毒，防火墙也缺乏防护的办法。

网络安全管理是一项实施企业安全性策略的系统工作，仅仅依靠身份验证、数据加密、访问控制与目录管理、防火墙等软硬件技术或网络通信技术不能完全解决问题，通常还必须配合以网络管理措施，还包括实施网络安全立法、安全评估以及实施网络安全行政管理制度。比如，网络管理员的素质很差，那么网络的安全性管理交替地极其危险。

7.7 网络管理初步

企业网络管理指的是一个企业网络的运行和网络资源的规划、设计、配置、监视、分析、组织和控制有关的所有必要活动，其主要任务是提高企业网络的工作性能和运行效益，保证企业网络的正常可靠运行，也包括能及时地分析与排除在网络中可能遇到的故障或者发现潜在的问题。

7．7．1 网络管理的基本功能

根据国际标准化组织所提出的 ISO7498-4"网络管理框架模型"中的观点，可以认为网络管理具有五大功能，即配置管理、失效与故障管理、性能管理、计费管理和安全管理。

所有的这些网络管理功能应在具体的网络管理标准协议的规范下予以保证，并由网络管理人员通过具体的网络管理平台和管理软件加以实施。

（1） 配置管理：这是网络管理的最基本功能之一。包括识别所管理的网络的拓扑结构（物理结构和逻辑结构），能标识出网络中的各个具体结点与对指定的设备与对象进行参数或状态的设置，组织并维护好网络内与配置有关的数据库，也可包括收集网络中的有关的信息等。

（2） 失效与故障管理：对网络中可能发生的失效或故障进行检测、诊断或定位，发现故障或失效时实施可以的隔离、纠正或网络重组，同时也包括在排除故障后对网络服务进行恢复。通常失效与故障管理依靠对网络的监测、检查与控制，网络管理过程中所提供的日志、警报与报告等手断也可用于失效与故障管理。

（3） 性能管理：性能管理的基本目标是通过评估网络中各类资源的运行状况，提供切实保证网络有效的通信能力和优化使用网络资源的相应办法与措施。为了实现性能管理的上述目标，需要在网络中测量或收集与网络性能有关的数据或信息，统计并分析当前与历史的数据，建立起性能分析的模型。

（4） 计费管理：本质上是监测与管理网络资源使用过程中以及通信传输所需要的费用与代价。

（5） 安全管理：对网络中设置的安全服务机制实施管理，保护网络资源与相应信息不被非法用户访问，对加密机制中的密钥也要加强管理。

7．7．2 网络管理协议

根据网络协议的两大体系结构（ISO/OSI 和 TCP/IP），相应地有着两个两个网络管理协议标准。一个是 ISO 和 ITU-T 联合制定网络管理标准 CMIS/CMIP 相应的网络管理标准框架是 ISO7498-4；另一个则是 TCP/IP 相应的 SNMP（simple network management protocol 简单的网络管理标准）。目前的网络管理产品大多数都支持 SNMP，SNMP 已成为事实上的工业标准。

概括起来说，SNMP V2/V3 的要点包括：

（1） 给出了受管理的网络的基本模型（SNMP 模型）及其网络管理工作方式。图 7.15 是 SNMP 模型的示意图。

一个受管理的网络由四部分组成：受管理的各个网络结点、一些网络管理工作站、网络管理信息、一个管理的协议（SNMP）。受管理的网络结点中运行一个 SNMP 代理进程（Agent），它维护本地的一个变量数据库，这些变量（在 SNMP 中称为对象）反映了相应结点的状态与运行历史，并可影响结点工作。网络管理工作站具体执行网络管理任务（也称为网络平台）其中有一个或多个管理进程，在网上与各代理进行通信交往，发出网管命令与获得响应信息。

（2） 网络管理信息在网络管理中起着十分重要的关键作用，最主要的网络管理住处称之为 MIB（管理信息库，BASE）。考虑到网络管理工作站必须能与来自不同厂商的各类设备（结点）交谈，必须硬性规定由所有结点设备所维护的信息库，使之符合规范。因此，在 SNMP 协议中祥细地描述了每一类代理必须维护的确切

信息以及指定的相应格式。SNMP 所能管理的所有基本对象的集合 MIB-II，其中定义了 175 个对象，分成 10 组。对定义的 175 个对象的每一个，给出了数据类型及其描述（即该变量用于什么目的）。事实上 SNMP 协议标准的主要篇幅都在描述 MIB，即定义什么设备应当跟踪哪些信息内容，这些信息应当如何实现通信交往。

在 MIB 中，也可允许加上厂商指定的一些变量。比如在交换式以太网中 RMON（远程监控）MIB 是交换式网络管理的关键，其中包含有许多组与原 SNMP 标准 MIB 不同的新的 MIB 统计数据。利用 RMON 及其有关的数据库 MIB，加上网络管理人员从远程监控测到远程端口信息量，可以立即获得全局网络的图景以及大量端口中的单个图景，因此要求所选设备或路由设备必须支持相应的 RMONN MIB，添购的主机及服务器网卡都应配有 SNMP 代程序和相应的 MIB。

（3） 给出了实施网络管理时采用的一种简单的协议（SNMP 协议）即网络管理工作站与结点中代理之间相互交往的具体方式（定义了七类可供发送的消息报文）大部分是采用请求-响应方式，由管理进程进行主动查询。偶尔也可由代理进行主动报告（称为自陷，比如出现了一件突发事件时）。

7．7．3 网络管理平台和网络管理软件

传统的基本网络管理大体上可采用以下一些相对简单的方式：

（1）使用网络操作系统或者网络系统中提供的实用程序或命令直接进行简单管理。比如：UNIX，Windows NT 和 Windows 98 等都提供有 PING 命令或实用程序，采用 PING＜IP 地址＞的方式可以测试是否能与对方连通。这类命令或实用程序一般可了解 TCP / IP 的激活、通信链路的连接等状况。发现不正常时，再进一步寻找原因。

（2）使用一些专用的诊断工具或测试仪器对网络进行测试与分析。可测试到网络中物理层、布线、媒体访问控制层等低层协议的实现情况。

（3）选用若干专门的网络管理应用程序进行专项监控与管理。比如：基于软件方式的规程分析工具软件、数据帧流量与结构分析的专门软件等。

对于 Intranet 来说，目前主要将采用网络管理平台和在该平台上开发出的相应网络管理应用软件进行全企业网范围内的有效网络管理，这些平台与软件都统一地支持开放式的 SNMP 网络管理协议标准。

网络管理平台是实现开放式集成化的网络集中统一管理的一项主要技术，提供了一种软件开发支持环境，在此环境中允许实现网络管理功能的集成，通常提供面向对象的 API（应用编程接口），有助于网络管理应用程序的开发或装人，也适应于接纳网络设备制造厂商所提供的相应网管软件。在开发网络管理应用程序时还可以利用网管平台提供的统一图形用户界面和 MIB 资源，从而可以提高开发效率和保证开发质量。

在企业网中网络管理平台一般使用一定性能与规模的通用性网络管理工作站主机，其主要任务包括：搜索与查找网络拓扑，构建、显示与维护企业网的网络拓扑图，构建有关信息的中心数据库，保护、维护与更新该数据库的内容，对网络中出现的各类事件进行监视、记录、分析、处理和过滤，提供友善的网络管理界面和使用方式等。企业网网络管理平台的代表产品有：IBM 公司的 Netview for AIX 或 TME10xA 公司的 Unicentre TNG，HP 公司的 Open View，Sun 公司的 Net Manager 和 Cabletron 公司的 Spectrum 等。相对而言，这些网管平台价格比较昂贵，有一定的网络运行开销。

通常，网络设备的主要供应厂商也都提供了相应的网络管理软件，比如：CISCO 公司的 Works，3Corn 公司的 Transcend / On demand，Bay Networks 公司的 Optivity 和 Cabletron 公司的 SPectrum 等。一般把这些由网络设备生产厂商所提供的网络管理软件称之为 LAN 网络管理级的软件，也允许把这些网络管理软件集成到上面所提到的各种分布式网管平台中去，作为基本的网络管理应用软件。

网络用户可以为本企业进一步开发或选购相应的网络管理应用软件，以适应企业网特定的应用需求。考虑到一个企业可能跨过 WAN 有多个 LAN（如有许多分部或办事处），实施这类网络的管理可采用分布式的策略，比如在各个分部的 LAN 中监控与收集相应的管理信息，并作必要的紧急事件处理，仅将有限的重要网络管理信息通过 WAN 向网管平台传递，可以节省网管的日常开支和提高网络管理效率。

第 8 章 数据结构基础

计算机算法的处理对象是描述客观事物属性的数据，由于客观事物的多样性，数据有不同的形式，如整数、实数、字符，以及所有计算机能够接收、存储和处理的符合集合。

数据结构是指数数据对象及其相互关系和构造方法，一个数据结构 B 可以用一个二元组表示为：B=（A，R）。其中，A 是数据结构中的数据（称为结点）的非空有限集合，R 是定义在 A 上的关系的非空有限集合。结构是指结点之间的关系，数据结构就是结点的有限集合关系有限集合。数据结构中，结点及结点间的相互关系是数据逻辑结构，数据结构在计算机中的存储内容，一般包括结点的值和结点之间的关系。数据结构的存储形式是数据的存储结构。

数据结构按逻辑关系不同的分为线性结构和非线性结构两大类，其中非线性结构又可分为树形结构和图结构，而树形结构又可分为树结构和二叉树结构。

算法和数据结构之间存在密切关系，，算法是建立在数据结构基础上的。明确了问题求解的算法，能较好地设计数据结构。但要设计好的算法，又常常依赖于合理的数据结构，数据结构是设计算法的基础。对于一些复杂的问题，常有因数据结构的差异问题的求解算法也会完全不同，设计合理的数据结构常可有效地简化算法。

8.1 线性表

数据结构中，线性结构习惯称为线性表，本节介绍线性表结构的基本运算及其应用程序实例。

线性表是最简单又是最常用的一种数据结构，线性表是由相同类型的结点组成有限序列。一个由 n 个结点心 $e_0, e_1, \cdots, e_{n-1}$ 组成的线性代表记：

$$（e_0, e_1, \cdots, e_{n-1}）$$

线性表的结点个数为线性表的长度，长度为 0 的线性代表称为空的线性表，简称空表。对于非空线性表，e_0 是线性表的最前一个结点，e_{n-1} 是线代表的最后一个结点。线代表的结点构成一个序列，对序列中两相邻结点 e_i 和 e_{i+1} 称 e_i 是 e_{i+1} 的前驱结点，e_{i+1} 是 e_i 的后继结点。其中 e_0 没有前驱结点，e_{n-1} 没有后继结点。

线性表最重要的性质是线性表中结点的相对位置是确定的。线性表中结点之间的关系由结点在线性表中位置所确定，用两相邻结点的偶。

（e_i 和 e_{i+1}）$(0 \leqq i \leqq n-2)$

表示两结点存在这种关系。如有两个线性表有相同的数据结点，但它们的结点在线性表中出现的顺序不同，则它们是两个不相同的线性表。

线性表的结点也称为表元，可记录，要求线性表的结点是同一类型的任何数据。线性表的结点可由若干个成分组成，其中能惟一标识表元的成分称为关键字，或简称键。为了讨论方便，往往只考虑表元的关键字，而忽略其成分。

1. 线性代表的基本运算

线性表包含的表元个数可以动态增加或减少，可以在任何位置插入或删除表元。线性表常用的运算可分成几类，每类有若干种运算。下面所列的其中一些常用的运算。

（1） 查找运算

- 查找线性表的第 I（$0 \leqq i \leqq n-1$）个表元；
- 在线性表中查找具有给定键值的表元；

（2） 插入运算

- 把新表元插在线性代表的第 I$(0 \leqq i \leqq n-1)$个表元；
- 把新表元插在具有给定键值的表元的前面或后面；

（3）删除运算

• 删除线性表的第 I（0≦i≦n-1）个表元；

• 删除线性表中具有给定键值的表元；

（4）其他运算

• 统计线性表中表元的个数；

• 输出线性表各表元的值；

• 复制线性表；

• 线性表分拆

• 线性表合并

• 线性表排序；

• 按某种规则整理线性表。

2. 线性表的存储

有多种存储方式能将线性表存储在计算机内，其中常用的顺序存储和链接存储。

（1） 线性表的顺序存储

线性表的顺序存储是最简单的存储方式。程序通常用一个足够大的数组，从数组的第一个元素开始，将线性表的结点依次存储在数组中，即线性表的第 I 个结点存储在数组的和第 I(0≦i≦n-1)个元素中，用数组元素的顺序存储来体现表中结点的先后次序关系。用数组存储线性表的最大优点是能直接访问线性表中的一个结点。

用数组存储线性表的缺点主要有两个：一是程序中的数组通常大小是固定的，可能会与线性表的结点可以任意增加或减少的要求相矛盾；二是执行线性表的结点插、删操作时要移动存于数组中的其他元素，使插和删操作不够简便。

（2） 线性表的链接存储

线性表链接存储是用链表存储线性表，最简单的是用单链表。如从链表的第一个表元开始，将线性表的结点依次存储在链表的各表元中。即线性表的第 I 个结点存储在链表的第 I（0≦i≦n-1）个表元中。链表的每个表元除要存储线性表结点的信息外，还要有一个成分用来存储其后继结点的指针。单链表就是通过链接指针来体现线性表中结点的和先后次序关系。每个链表还要有一个指向链表第一表元的头指针。当链表为空时，头指针为空值；当链表非空时，头指针指向链表的第一个表元，链表的最末一个表元的后继指针为空。用链表存储线性工表的优点是线性表的每个表元的实际存储位置可以任意，这给线性表的插、删操作带来

方便，只要改变链表有关表元的后继指针就能完成插或删的操作，不需移动任何表元。

其缺点也主要有两条：一是每个表元增加了一个后继指针成分，要花费更多的存储空间；二是不便随机直接访问线性表的任一结点。

3. 线性表上的查找

线性表上的查找运算是指在线性表中找某个键值的结点。根据线性表中的存储形式和线性表本身的性质差异，有多种查找算法，如：顺序查找、二分法查找、分块查、散列查找等。

4. 线性表的新结点插入

（1） 顺序存储线性表的插入

设线性表结点的类型为整型，插入之前有 n 个结点，把值为 x 的新结点插在线性表的第 I(0≦i≦n)个位置上。完成插入主要有以下步骤：

• 检查插入要求的有关参数的合理性；

• 把原来的第 n-1 个结点至第 I 个结点依次往后移一个数组元素位置；

• 把新结点放在第 I 个位置上；

• 修正线性表的结点个数；

实现插入算法的函数定义如下：

[函数]顺序存储线性表的插入。

```
int sp_insert(list, npt, I, maxn, x)
int * list, / * 线性表首结点指针 * /
        * npt, / *记录线性表结点数的变量的指针 * /
        I, / * 表元插入位置 */
         maxn, / * 存储线性表的数组元素个数 * /
        x;  / * 插入结点值 * /
{int k;
        if(I<0||I>npt)return 1; / * 不合理的插入位置 I * /
        if(* npt ==maxn )return 2; / * 存储线性代表的数组已满 * /
                                    / * 第 n-1 结点至第 I 结点依次往后
移一个位置 * /
        for(k=* npt; k>I;k--)list[k]=list[k-1];
        list[i]=x; / * 新结点插入 * /
        ++ * npt; / * 线性表结点数增 1 * /
```

```
    return 0; / * 插入成功 * /
}
```

设有变量定义：

```
# define N 100
int a[N],n;
```

用数组 a[]存储一个线性表，用 n 记录线性表的结点个数，将值 x 插入线性表的第 I(0≤ i≤n)个位置上，可用如下 C 代码调用函 sp_insert()：

```
flg=sq_inser(a, &n, I, N, x);
```

仅当函数返加值为 0 时，插入才是成功的。

在具有 n 个结点的线性表上插入新结点，其时间主要花费在移动结点的循环上。若插入任一位置的概率相等，则在顺序存储性代表中插入一个新结点，平均需要移动线性表的一半结点。

（2） 链接存储线性表的插入

设链表结点类型的定义为：

```
typedef struct node{
     int data;
     struct node * link;
}NODE;
```

在链接存储线性表中插入一个键值为 x 的新结点，分以下 4 种情况：

• 在某指针 p 所指结点之后插入；

• 插在首结点之前，使新结点成为新的首结点；

• 接在线性表的末尾 ；

• 在有序链接存储线性表中插入，使新的线性表依旧有序。

其中在有序链接存储线性表上的插入运算包含寻找插入点的步骤，找到插入点后又要按上述前两种情况分别作插入处理。将上述 4 种情况的插入运算分别编写成函数如下：

[函数]链接存储性表上，在某指针所指结点之后插入键值为 x 的新结点。

```
void lq_insl(p,x)
NODE * p; /* 新结点插在 p 所指结点之后 * /
```

```
int x; / * 新结点的键值 * /
{ NODE * u;
    u=(NODE * )malloc (sizeof (NODE));
    u->data=x;
    u->link=p->link;
    p->link=u;
}
```

因其余三个插入操作都有可能要更改链表的首指针，所以它们的函数中关于链表头指针的参数应为链表头指针的指针

[函数]链接存储线性表上，在首结点之前插入键值为 x 的新结点。

```
void lq_ins2(hpt, x)
NODE ** hpt; / * 链表头指针的指针 * /
int x; / *新结点的键值 * /
{  NODE *u;
    u=(NODE *)malloc(sizeof (NODE));
    u->data=x;
    u->link= *hpt;
    * hpt=u;
}
```

[函数]在链接存储性表的末尾接上一个键值为 x 的新结点。

```
void lq_ins3(hpt, x)
NODE * * hpt; / * 链表头指针的指针 * /
int x; / * 新结点的键值 * /
{   NODE * u, * p;
     u=(NODE * )malloc(sizeof(NODE));
     u->data=x;
     u->link=NULL;
     if(* hpt==NULL){ / * 如链表为空链表 * /
       * hpt=u;
return;
}
/ * 从链表首结点开始顺序走向末尾结点 * /
for (p= * hpt;p->link!=NULL;p=p->link);
p->link=u;
}
```

对于函数 lq_ins3(),如有链表引入末尾结点指针，函数就可以省去寻找链表末指点的步骤。

[函数]在有序链接存储线性表中插入一个键值为 x 的新结点。

```
void lq_ins4(hpt,x) / *设链表从小到大有序 * /
NODE * * hpt; / * 链表头指针的指针 * /
```

```
int x; / * 新结点的键值 * /
{ NODE u, * q, * p;
    u=(NODE * )malloc(sizeof(NODE));
    u->data=x;
     / * 从链表首结点开始顺序寻找 * /
    for(p= *hpt;p && p->data<x;q=p,p=p->link);
     if(p==* hpt)*hpt=u;
     else q->link=u;
     u->link=p
}
```

5. 线性表的结点删除

（1） 顺序存储线性表的结点删除

顺序存储线性表的删除运算是在有 n 个表元的线性表中，删除第 I(0≦i≦n-1)个表元。删除时应将第 I+1 个表元至第 n-1 个表元依次向前移一个数组元素位置，共移动 n-I-1 个表元。完成删除主要有以下几个步骤：

- 检查删除要求的有关参数的合理性；
- 把原来第 I+1 个表元至第 n-1 个表元依次向前移一个数组元素位置；
- 修正性表表元个数。

实现删除算法的函数定义如下：

[函数]顺序存储线性表的删除运算。

```
int sq_delete(list,npt,I)
int * list, / * 线性表首表元指针 * /
        * npt, / * 记录线性表元元数的变量的指针 */
        I; / * 删除表元 * /
{ int k;
        if(I<0||I>=* npt) return 1; / * 不合理的删除位置 I * /
        if( * npt ==0)return 2; / *线性表已空 * /
             / * 第 I+1 表元至第 n-1 表元依次向前移一个位置 * /
        for(k=I+1;k< * npt; k++)list[k-1]=list[k];
        --* npt; / * 线性表表元数减 1 * /
       return 0; / * 删除成功 * /
}
```

用数组 a[]存储一个线性表，用 n 记录线性表的表元个数删除线性表的第 I(0≦i≦n)个表元，可用如下 C 代码调用函数 sq_deleted()：

```
flg=sq_delete(a, &n, i);
```

仅当函数返回值为 0 时，删除才是成功的。

在具有 n 个表元的线性表上删除表元，其时间主要花费在移动表元的循环上。若删除任一表元的概率相等，则在顺序存储线性表中删除一个表元，平均需要移动线性表的一半表元。

（2） 链接存储线性表的删除

对于链接存储线性表上删除指定值结点的删除运算。需考虑几种情况，一是链表为空链表，不执行删除操作；二是要删除的结点恰为链表的首结点，应将链表头指针改为指和原首结点的后继指点；其他情况，程序先要在链表中寻找指定值的结点，从链表首结点开始顺序寻找。若找到，执行删除操作，若直至链表末尾没有指定值的结点，则不实行删除操作。完成删除可有友下几个步骤组成：

- 如链表为空链表，则不执行删除操作；
- 若链表的首结点的值为指定值，更改链表的头指针为向首结点的后继结点；
- 在链表中找指定值的结点；
- 将找到的结点删除。

因函数要更改链表的首指针，所以关于链表头指针的参数应设定为链表指针。根据上述计算步骤写成函数如下：

[函数]链接存储线性表的删除

```
int lq_delete(hpt,a)
struct node * * hpt; / * 链表头指针的指针 * /
int a; / * 要删除的结点键值 * /
 { struct node * p, * q;
    if(q= * hp)==NULL )return 1; / * 链表已为空链表 * /
    if(q->data==a){ / *要删除链表的首结点 * /
    * hpt=q->link; / * 更改链表头指针 * /
    free(q); / * 释放被删除结点的空间 * /
    return 0; / * 删除成功 * /
}
  for(;q->data!=a && q->link!=NULL;p=q;q=q->link);/ * 寻找 * /
   if(q->data==a){ / * 找到 * /
         p->link=q->link; / * 被删除的结点从链表脱钩 * /
        free(q); / * 释放被删除结点的空间 * /
        return 0; / * 删除成功 * /
     }
```

```
      return 1; / * 没有指定值的结点，未执行删除操作 * /
}
```

要删除以 head 为头指针的链表中值为 a 的结点可用以下 C 代码来实现：

```
flg=lq_delete(&head, a)
```

仅当函数返回值为 0 时，值为 a 的结点才被真正删除。

[例 8. 1]链接存储线性表的创建、插入新表元、删除表元、线性表颠倒示意程序。

下面的程序以整数链表为例，提供一组菜单：完成链接存储性表的创建、向链表插入一个新表元、删除指定值的表元、链接存储线性表颠倒等操作。用户输入选择，程序完成相应的工作。

[例 8. 1 程序]

```
#include<stdio.h>
#include<stdlib.h>
struct inNode{
    int value;
    struct inNode * next;
};
struct inNode * head;
/ * 能实现遍历链表功能的函数定义如下： * /
void traveLink(struct inNode * h)
{   struct intNode * p=h;
     while(p!=NULL){
     printf("%4k",p->value); / * 输出表元的值 * /
     p=p->next; / * 准备访问下一个表元 * /
 }
 printf("\n");
}
/ * 无序链表上的动态查找 * /
struct intNode * searchDSLink(struct intNode * h,int key,intNode ** pp)
{  struct int Node * v=h, *u=NULL;
    while((v!=NULL)&&(v->value!=key)){
         u=v; / * 准备考察下一个表元，当前表元成为后继表元的前驱表元
* ; /
         v=v->next; / * 考察下一个表元，后继表元成为当前表元 * /
}
    *pp=u; return v;
}
struct intNode * createList()
```

```
{  struct intNode * h, / * 链表的头指针 * /
     * tail, / * 链表末尾表元的指针 * /
     * p;
     int n; char c;
     h=tail=NULL;
     printf('Input data .\n");
     while(scanf("%d",&n)>0){ / * 有整数读入 * /
          p=(struct intNode * )malloc(sizeof(struct intNode));
          p->value=n;
          p->next=NULL;
          if(h==NULL)h=tail=p;
               else talil=tail->next=p;
     }
     do{scanf("%c",& c);
     }while(c!=\n); / * 掠过整数序列之后，回车之前的字符 * /
     return h;
}
struct inNode * reverse(sturct inNode * h)
{   stuct inNode * p,*v1,* v2;
     v2=h; / *v2 指向链表的首表元 * /
     v1=NULL; / *开始颠倒时，已颠倒部分为空 * /
     while(v2!=NULL){ / *还末颠倒完，循环 * /
     p=v2->next;
     v2->next=v1;
     v1=v2;
     v2=p;
     }
     return v1;
}
struct inNode * inNode (void)
{   int d; struct intNode * p;
     printf("输入插入表元的值");
     scanf("%d",&d);
     p=(struct inNode * )malloc (sizeof(sturct inNode));
     p->value=d;
     p->next=head;
     return p;
}
struct intNode * delNode(void)
{   int d; struct intNode * p,*q;
     printf("输入删除表元的值")'
     scanf("%d",& d);
     p=searchDSLink(head,d,&q);
     if(p==NULL)printf(" 没有指定值%d 的表元。、n",d );
```

```
     else{
          if(q==NULL)head=p->next;
          else q->next=p->next;
          p->next=NULL;
          free(p);
     }
     return head;
}
struct intNode * revNode()
{
     return reverse(head);
}
char * menuName[]={"创建一个整数链表",
                                        "向链表插入一个新结点",
                                        "从链表删除一个结点",
                                        "将链表颠倒",
                                        ""};
struct intNode * (* fpt[])(void)=
{createList ,inNode,delNode, revNode, NULL};
struct intNode * (* menu(void)) / * 函数返回函数的指针 * /
{   int ans,k;
     printf("请选择以下菜单命令。\n");
     for(k=0;menuName[k][0]!='\0';k++)
          printf("\t%d:%s\n",k+1,menuName[k]);
     printf("\t 其他选择结束程序运行。\n");
     scanf("%d",&ans);
     if(ans<1||ans>k)return NULL;
     return fpt[ans-1]; / * 返回函数的指针 * /
}
void main()
{   struct intNode * (* fp)(void);
     head=NULL;
          while(1){
               if(fp=menu())==NULL)break;
               head=(* fp)();
               travelink(head);
     }
}
```

8.2 栈

栈是只允许在同一端进行插入和删除运算的线性表，见图 8.1（a）。允许插入和删除的那一端称为栈顶，另一端为栈底。若有栈

$$S=(s_0, s_1, \cdots, s_{n-1})$$

则 s_0 为栈底结点，s_{n-1} 为栈顶结点，习惯称栈的结点插入为进栈，栈的结点删除为出栈。因为最后进栈的结点必定最先出栈，所以栈具有后进先出的特征。

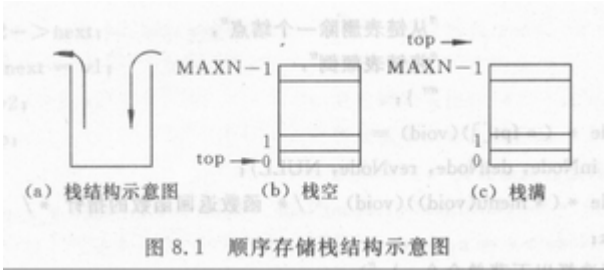


图 8.1 顺序存储栈结构示意图

1. 顺序存储栈

可以用顺序存储线性表来表示栈，为了指明当前执行插入删除运算的栈顶位置，需要一个游标变量 top（习惯称为栈指针，注意与指针变量的区别）， top 指出栈顶表元在数组中的下标。按 top 所指位置不同，用数组表示栈一般有两种不同形式，这里给出其中的一种。
top 是下一次进栈时，进栈表元存入的数组元素下标，当栈为空时，top=0;栈满时，top=MAXN（数组元素个数），见图 8.1 的（b）和（c）。表元进栈时，首先把进栈表元存入以 top 为下标的数组元素中，然后 top 加 1 使 top 变为下一次进栈表元在数组中的下标。表元出栈时,首先执行 top 减 1,然后把以 top 为下标的栈顶表元送到接受出栈表元的变量中。同样限制在栈空情况下，不能再出栈；在栈满足情况下，不能再进栈。下面是相应的进栈和出栈运算的函数定义：

[函数]顺序存储栈的进栈函数。

```
int push(NODE stack[],int maxn,int * toppt, NODE x) / * 进栈函数 * /
/ * 设栈点的数据类型为 NODE，栈空间最多能存储 maxn 个结点 * /
{   if(* toppt>=maxn) return 1; / * 栈满，进栈失败返回 1 * /
     stack[* toppt]=x; / * 完成进栈运算 * /
     ++ * toppt
     return 0; / * 进栈成功，返回 0 * /
}
```

[函数]顺序存储栈的出栈函数

```
int pop(NODE stack[],int * toppt ,NODE * cp) / * 出栈函数 * /
{   if(* toppt ==0)      return 1; / * 栈空，出栈失败返回 1 * /
     * toppt;
     * cp=stack[* toppt]; / * 完成出栈运算 * /
     return 0; / * 出栈成功，返回 0 * /
}
```

2.链接存储栈

栈也可以用链表实现，用链表实现的栈称为链接栈。链表的第一个表元是栈顶结点，链表的末尾表元是栈底结点，链表的首表元指针就是栈项指针 top,top 为 NULL 的空链表是空栈。设栈结点类型为：

```
typedef int NODE; / * 设栈元值是某种 NODE 类型的，这里暂定灰是整型的。 * /
typedef struct node {
     NODE data; / * 栈元值，某种 NODE 类型 * /
     struct node * link;
}LNODE;
```

利用有关链表知识，直接写出链接栈的进栈和出栈函数如下：

[函数]链接存储栈的进栈函数。

```
void l_push(NODE x, LNODE * * toppt)
/ * 设栈结点的数据类型为 NODE * /
{   LNODE * p=(LNODE * )malloc(sizeof(LNODE));
     p->data=x;
     p->link= * toppt;
     * toppt=p;
}
```

[函数]链接存储栈的出栈函数

```
int l_pop(NODE * cp, LNODE * * toppt)
{   LNODE * p=* toppt;
     if(* toppt==NULL)return 1; / * 空栈 * /
     cp=p->data;
     * toppt=p->link;
     free(p);
     return 0;
}
```

3．栈实用实例

栈在计算机语言处理和将递归算法改写成非递归等方面起着非常重要的作用。现举这两方面的实例。

[例 8．2]计算数字表达式的值

设数字表达式包含的运算符有加（+）、减（-）、乘（*）、除（/）和乘幂（^），其中乘幂运算符的优先级最高，且有连续乘幂运算时，自左向右的运算。乘和除

的运算优先级比加减运算的优先级高，但优先级相同的加减乘除四则运算自左至右运算。另可用括号改变运算的优先级，括号内的运算优先。

另约定一行一个数字表达式，加车符作为一个表达式的结束符。

为利用栈实现表达式计算，程序引入两个栈，即运算分量栈和运算符栈。程序在对表达式求值时，按自左至右顺序扫描数字表达式字符行，当遇到数字时，将数进运算分量栈；遇运算符（包括括号）时，都要用它的优先级（栈外级）和运算符栈顶运算符的优先级（栈内级）进行比较，栈外优先级高，则进栈；反之，则出栈处理，直到运算符栈顶运算符的优先级较低为止，再将原栈外运算符进栈。所以，只要对所有运算符的栈内外优先级作恰当的定义，即可实现表达式的自动求值。这里引入运算符的栈内和栈外优先级，是因为运算符自右向左运算，也有运算符处左向右运算，仅仅是为了统一算法，方便处理。表达式求值的大致算法思想如下：
初始时，在运算符栈顶置″（″。

（1） 凡遇运算分量一律进运算分量栈；

（2） 遇到运算符，则按以下情况分别处理：

栈顶运算符的栈内级比当前栈外运算符的栈外级低，则栈外运算符进栈；

当栈顶运算符的栈内级比当前栈外运算符的栈外级高，并且栈顶运算符是计算运算符时，则出栈处理。即，将运算分量出栈，运算符出栈，并计算，再将计算结果出栈。直到栈顶运算符的栈内级比当前栈外运算符的栈外级低为止。然后再按栈外运算符的种类分以下三种情况：

·闭括号″）″，在正常情况下，运算符栈顶运算符应为″（″，删去栈中的″（″；

·回车符，这是表达式的结束符。在正常情况下，运算符栈顶运算符为初始设置的″（″，运算分量栈顶结点值即是所求表达式之值；

·其他运算符，将该运算符进栈。

限于篇幅，以下程序未检查表达式的单词出现顺序的合理性，如程序对表达式的每个单词的后继单词的合理性要求能做检查，程序就会更好。细节见以下程序。

[例 8．2 程序]

```
#include <stdio.h>
#include <stdlid.h>
#include<math.h>
#define POW 1
#define MUL 2
#define DIV 3
```

```
#define ADD 4
#define SUB 5
#define Lp 6
#define Rp 7
#define END 8
#define Epsilon le-7
typedef double NODE;
struct {
          char op;
          int code;
}opch Tbl[]={{'* ',2},{'/',3},{'+',4},{'-',5},{'(',6},{'}',6),
               {')',7},{'^',1},{'\n',8},{'',-1}};
typedef struct node{
     NODE data;
     struct node * link;
}LNODE;
LNODE * optop, * numtop;
/ * [函数]链接存储栈的进栈函数 * /
void l_push(NODE x,LNODE **toppt)
{   LNODE * p=(LNODE *)malloc (sizeof(LNODE));
     p->data=x;
     p->link=* toppt;
     toppt=p;
}
/ * [函数]链接存储栈的出栈函数 * /
int l_pop(NODE * cp,LNODE ** toppt)
{   LNODE * p=* toppt;
     if(* toppt==NULL)return 1; / * 空栈 * /
     *  cp=p->data;
     * toppt=p->link;
     free(p);
     return 0;
}
int osp[]={5,3,3,2,2,5,1,1,} / * ,^,* ,/ ,+,-,(,),"的栈外优先级 * /
     isp[]={4,3,3,2,2,0}; / *, ^,*,/,+,-,(的栈内优先级 * /
void synError()
{   double temp;
     printf (" 表达式句法错!");
     while(optop!=NULL)l_pop(&temp,&optop);
     while(numtop!=NULLl_pop(&temp,&numtop);
     exit(0);
}
double eval(char tag,double left,double right)
{   int n;
```

```
     double result;
     switch (tag){
          case POW:for(n=1,result=left;n<right;n++)
                    result * =left;
               return result;
          case ADD;return left+right;
          case SUB: return left-right;
          case MUL:return left * right;
          case DIV:if(fabs(right)<=Epsilo){
                    printf("除 0 出错!\n");
                    exit(1);
               }
               return left/ right;
     }
     printf("表达式有错!\n" );return 1.0;
}
int c='';
#define RADIX 10
int getToken(double * nump)
{   double dradix,num;
     int I;
     while(c==''||c=='\t')c=getchar(); / * 掠过空白符 * /
     if(c<'0'||c>'9'){
          for(I=0;opchTbl[i].code!=-1&7opchTbl[i].op!=c;I++);
          if(opchTbl[i].code==-1)synError(); / * 非法字符 * /
          if(c!='\n')c=getchar(); / * c 中存储下一个字符 * /
          return opchTbl[i].code; / * 返回运算符的内部码 * /
     }
     num=0.0 / * 遇到数字,翻译数 * /
     while(c>='0'&&c<='9'){
          num=RADIX * num+c-'0';
          c=gerchar();
     }
     if(c=='.'){
          dradix=1.0/ RADIX;
          c=getchar();
          while(c>='0'&& c<='9'){
               num=num+(c-'0')* dradix;
               dradix/=RADIX;
               c=getchar();
          }
     }
     * nump=num;
     return 0;
```

```
}
void main()
{   double num,dop,operand1,operand2,res;
     int type;
     char ans, op;
     do{
          printf("请输入表达式!\n");
          optop=numtop=NULL;
          l_push(Lp,&optop); / * 左括号进栈 * /
          do c=getchar(); / * 掠过前导空白类字符 * /
          while(c==''||c=='\n'||c=='\t');
          while(1){
               type=getToken(&num);
               if(type==0)l_push(num,&numtop); / * 数字进运算分量栈 *
/
               else{
                         if(osp[type-1]>isp[(int)optop->data-1])
                             l_push((double)type,&optop);
                         else{
                                 while(osp[type-1]<=isp[(int)optop->da
ta-1]
                                     &&optop->data<=5{
                                 if(l_pop(&dop,&optop)) / * 运算符出栈
* /
                                     synErroor();
                                 op=(char)dop;
                                 if(l_pop(&operand2,&numtop)) / * 运算
分量 2 出栈 * /
                                     synError();
                                 if(l_pop(&operand1,&numtop) / * 运算分
量 1 出栈 * /
                                     synError();
                              res=eval(op,operand1,operand2); / * 双目运
算 * /
                              l_push(res,&numtop); / * 运算结果进栈 * /
                }
                if(type==END)break; / * 一个表达式计算结束 * /
                if(type==Rp){ / * 将栈中'('退去 * /
                    do if(l_pop(&dop,&optop))
                        synError(); / * 栈空出错 * /
                    while((char)dop!=Lp); / * 连续退栈至'(' * /
                }else
                    l_push((double)type,&optop;) / * 栈外运算符进栈 *
/
```

```
            }
          }
      }
    if(l_pop(&operand1,&numtop))synError(), / * 计算结果出栈 * /
     while(optop!=NULL) l_pop(&dop,&optop);
     while(numtop!=NULL) l_pop(&res,&numtop)
     printf("结果为%f\n",operand1);
     printf("继续吗?");
     scanf("%c",&ans);
   }while(ans=='y'||ans=='Y');
}
```

在上述程序中,将运算栈的栈元值类型也采用 double,是为了与运算分量栈共用进栈和出栈函数.

[例 8.3]用栈改递归为非递归,计算 Ackermann 函数 A(n,x,y).

栈的另一个典型应用是用栈将递归计算改为非递归计算.对于那些计算过程具有递归性,或算法的操作对象的数据结构是递归定义的情况,如树形数据结构那样,这类算法特别适宜用递归形式来构造.递归算法具有结构清晰,易读易理解,并且其正确性也容易验证.因此,将递归算法编写成递归函数给程序的编写和程序调试都会带来很大的方便.但递归函数在运行过程中会耗费太多的时间和空间,特别对于使用不支持递归程序语言的情况,为了追求算法的时空效率,递归算法常借助栈将其转化为非递归.

将递归算法转化为非递归,需要一个工作栈,它有两个作用:一是将递归调用时的实在参数和函数返回地址传递给下一层执行的递归函数;二是保存本层的参数和局部变量,以便从下一层返回时重新使用它们.

下面以著名递归函数 Ackermann 函数为例说明如何用工作栈将递归转化为非递归.

Ackermann 函数定义如下，其中，n,x,y 均为非负整数：

A（n,x,y）=x+1 n=.0
=x n=1 且 y=0
=0 n=2 且 y=.0
=1 n=3 且 y=0
=2 n≥4 且 y=0
=A（n-1）,A(n,x,y-1),x)n≠0 且 y≠0

先举一个实例看 Ackermann 函数是如何计算的。

A（3，1，2）=A（2，A（3，1，1，），1）
=A（2，A（2，A（3，1，0），1），1）

```
                        =A（2，A（2，1，1），1）      / * A（3，1，0）
=1 */
                        =A（2，A（1，A（2，1，0），1），1）
                        =A（2，A（1，0，1），1）      / * A（2，1，0）
=0 * /
                        =A（2，A（0，A（1，0，0），0），1）
                        =A（2，A（0，0，0），1）      / * A（1，0，0）
=0 * /
                        =A（2，1，1）          / * A（0，0，0）=1 * /
                        =A（1，A（2，1，0），1）
                        =A（1，0，1）              / * A（2，1，0）=0 *
/
                        =A（0，A（1，0，0），0）
                        =A（0，0，0）              / * A（1，0，0）=0 *
/
                        =1                  / * A（0，0，0）=1 * /
```

由上述计算过程可以看出，计算 A（n,x,y）,当 n!=0,y!=0 时，用函数调用 A（n-1,B,x）代替，其中 B=A(n,x,y-1).在计算 A(n,x,y-1)前，应先将参数 n,x,y 保存起来,待计算出 A(n,x,y-1)的值之后,设其值为 B,再回来计算 A(n-1,B,x).如果计算出 A（n,x,y-1）时，仍有 y-1>0,时，仍有 y-1>0,又要将 n,x,y-1 保存起来，去计算 A（n,x,y-2），…，直到计算出值，再返回来计算各中间值。反复进行这一过程，直到最后求得最初的函数值。

为此，若不用递归计算，可用一个栈来保存这三个参数，以便不断时行递归和返回。用栈计算 A（n,x,y）的算法如下；

[算法]用栈计算 Ackerann 函数的函数 A（n,x,y）

```
double A(n,x,y)
{      将结构（n,x,y）进栈；
       do{ / * 循环计算至栈空 * /
          tnode=栈顶结点；
          while(tnode.n !=0&&tnode. y!=0){ / * 递归计算参数进栈 * /
                 修改栈顶结点值；
                 形成新结点，并进栈；
             }
          退栈，并将取得的栈顶点的值存入 tnode;
          计算：B=A（tnode）；
          if（栈非空）将计算结果 B 回填到栈顶点的 x；
       }while(栈非空)；
     返回 B;
}
```

按上述算法编写的程序如下：

[例 8. 3 程序]

```
#include<stdio.h>
#define MAXN 8000
typedef struct node{ / * 栈元类型 * /
    int n;
    double x,y;
}  NODE;
NODE stack[MAXN];
int top,deb,depth;
int push(NODE stack[],int maxn, int * toppt,NODE x) / * 进栈函数 * /
{    if(* toppt>=maxn)
            return 1; / * 栈满，进栈失败返回 1 */
        stack[* toppt]=x; / * 完成进栈运算 * /
        ++ * toppt;
        if(* toppt>depth)    depth=*toppt;
        return 0; / * 进栈成功，返回 0 * /
}
/ * [函数]顺序存储栈的出栈函数 */
int pop(NODE stack[],int * toppt, NODE * cp) / * 出栈函数 * /
{    if(* toppt==0)return 1; / * 栈空，出栈失败返回 1 * /
        --* toppt;
        * cp=stack[* toppt]; / * 完成栈运算 * /
        return 0; / * 出栈成功，返回 0 * /
}
double A(NODE node)
{    int k;double B;
        NODE tnode;
        top=0;depth=0;
        push(stack.MAXN, &top,node); / * 将 node 进栈 * /
        do{ / * 循环计算至栈空 * /
            tnode=stack[top-1]; / * 读栈顶结点的值的存入 tnode * /
            while(tnode.n!=0&&tnode.y!=0){/* 递推，将计算参数进栈 */
            / *修改栈顶点的参数，并求得新结点进栈 * /
                stack[top-1].n--;stack[top-1].y=stack[top-1].x;
                stack[top-1].x=-1.0; / * 表示 x 值不定 * /
                tnode.y-=1.0;
                if(push(stack,MAXN,& top,tnode))
                return-1.0; / * 将 tmode 进栈 * /
            }
        if(deb){
            for(k=0<top-10?top-10:0;k<top;k++)
                printf("N=%4d,x=%8f,y=%8f\n",
                    stack[k].n,stack[k].x,stack[k].y);
```

```
            scanf("% * c");
        }
            / * 退栈，并将取得的栈顶点的值存入 tnode * /
            pop(stack,&top,&tnode);
            / * 计算：B=A（tnode） * /
        if(tnode.n==0)B=tnode.x+1.0;
        else if(tnode,n==1)B=tnode.x;
        else if(tnode.n==2)B=0.0;
        else B=2.0;
        if(top) / * 栈非空将栈顶点的 x 值改为 B * /
                stack[top-1].x=B;
    }while(top); / * 栈非空 * /
    return B; / * 返回 B * /
  }
void main()
{     NODE node1={3,1,2},node2={3,2,5},node3={4,3,2};
        printf(" DEB? ");scanf("%d",&deb);
        printf("A(3,1,2)=%f",A(node1));
        printf("\t%d\n",depth);
        scanf("% * c");
        printf("A(3,2,5)=%f",A(node2));
        printf("\t%d\n",depth);
        scanf("% * c");
        printf("A(4,3,2)=%f",A(node3));
        printf("\t%d\n",depth);
}
```

栈的应用例子很多，如用于回溯法找解。以走迷宫为例，从迷宫入口开始，沿有通路的方向前进，一旦走不通时，就要从原路退回，再在未走过的方向去寻找可走的道路，若有未走过的通路，则继续向前走，否则继续回退。为能正确返回和计算，把走过的路线和方向都记录在栈中，就能实现非递归，回溯找解。

8.3 队列

队列是只允许在一端进入插入，另一端进行删除运算的线性表。见图 8.2，允许删除运算的那一端称为队首，允许插入运算的另一端为队尾。习惯称队列的结点插入为进队，队列的结点删除为出队。若有队列：

$$Q=(q_0,q_1\cdots,q_{n-1})$$

则 q_0 为队首结点，q_{n-1} 为队尾结点。因最先进入队列的结点将最先出队，所

以队列具有先进先出的特性。

1. 顺序存储队列

可以用顺序存储线性表来表示队列，为了指明当前执行出队运算的队首位置，需要一个游标变量 head（习惯称为头指针），另为了指明当前执行进队运算的队尾位置，需要一个游标变量 tail（习惯称为尾指针）。

开始时，让队列 head 和 tail 都指向数组的前端，是一个空队列。在队列未满情况下，队列可执行进队运算。进队时，首先将新结点送到以 tail 为下标的数组元素中，然后让 tail 加 1，变为指向队列新结点的存放下标。队列非空时，可执行出队运算。出队时，首先把 head 所指的队列首结点送到接受结点的变量中，然后让 head 加 1，使 head 指向新的队首结点。当队列的头指针 head 超过尾指针 tail 时，就出现队空状态。

若用有 MAXN 个元素的数组表示队列，随着一系列进队和出队运算，队列的结点移向存放队列的数组的尾端，会出现数组的前端而空着，而队列空间已用完的情况。一种可行的解决办法是当发生这样的情况时,把队列中的结点移到数组的前端,前修改头指针和尾指针.另一种更好的解决办法是采用环形队列.

环形队列就是将实现队列的数组 q[]的首元与末元 q[MAXN-1]连接起来.队空的初态为 head=tail=0.在环形队列中,当 tail 赶上 head 时,队列满.反之,当 head 赶上 tail 时,队列变为空.这样队空和队满的条件都同为 head=tail,这给程序判别队空或队满带来不便.为此采用当队列只剩下一个空闲结点的空间时,就认为队列已满的简单办法,以区别队空和队满.进队和队运算描述如下:

[算法]环形顺序存储队列进队

```
{    if(队满) / * tail 即赶上 head,只剩下一个空闲的结点空间 * /
           进队失败;
       else 执行进队运算;
        修正 tail; / * 让 tail 加 1 并模 MAXN * /
}
```

[算法]环形顺序存储队列出队

```
{    if(队空)          / * head 已赶上 tail * /
           出队失败;
       else{执行出队运算;
           修正 head; / * 让 head 加 1 并模 MAXN * /
    }
}
```

环形顺序的进队和出队函数如下:

[函数]环形顺序存储队列进队函数

```
int en_c_queue(NODE q[],int maxn,int * tpt,int * hpt,NODE x)
/* 队列结点数据类型为 NODE,用能存储 maxn 个结点的数组实现 * /
```

```
{    if((*tpt+1)%MAXN==*hpt / * 队满 * /
           return 1;
      q[*tpt]=x;
       * tpt=(*tpt+1)%MAXN;
       return 0;
}
```

[函数]环形顺序存储队列出队函数

```
int de_c_queue(NODE q[],int maxn, int * tpt, int * hpt,NODE * cp)
{    if(* hpt==* tpt)return1; / * 队空 * /
     * cp=q[* hpt];
     * hpt=(* hpt+1)%MAXN;
     return 0;
}
```

2.链接存储队列

队列也可以用链接存储线性表实现,用链接存储线性表链表实现的队列称为链接队列.链表的第一个表元是队列首结点,链表的末尾表元是队列的队尾结点,队尾结点的链接指针值为 NULL.队列的头指针 head 指向队首结点,队列的尾指针 tail 指向队尾结点.当队列的头指针 head 值为 NULL 时,队列为空.直接利用有关链表的知识,可写出链接队列的进队和出队的函数.设有:

```
typedef struct node{ / * 队元类型 * /
    NODE data;
    struct node * link;
}LNODE:
```

[函数]链接队列进队函数

```
void len_queue(LNODE * * hpt, LNODE ** tpt, NODE x)
{    LNODE * p=(LNODE *)malloc (sizeof(LNODE));
     p->data=x;
     p->link=NULL;
     if(* hpt==NULL)* tpt= * hpt =p;
     else
        * tpt=(*tpt)->link=p;
}
```

[函数]链接队列出队函数

```
int lde_queue(LNODE * *hpt,LNODE * * tpt,NODE * cp)
{    LNODE p =* hpt;
     if(p==NULL)    return 1; / * 队空 * /
```

```
        * cp=p->data;
        * hpt=p->link;
        if(* hpt==NULL)* tpt=NULL;
        free(p);
        return 0;
}
```

3.队列应用实例

队列数据结构常用于顺序处理情况,如操作系统中的各种资源请求队列,计算机事件模拟处理等.以下以事件模拟处理为例讨论的典型应用.

[例 8.4]某公司有一个顾客接待站,在单位时间内随机地有顾客到接待站询问.设每位顾客的接待时间是某个范围内的随机值.设接待站只有一名接待人员,要求程序模拟统计在设定时间内,接待员的总空闲时间和顾客的平均等待时间.假定模拟数据已按顾客到达接待站的先后顺序依次存于某个正文数据文件中.对应每位顾客有两个数据,到达时间和需要接待的时间.

在计算机模拟程序中,程序按模拟环境中的事件出现顺序逐一处理事件.当一个事件结束时,下一事件结束时,下一事件隔一段时间才发生,则程序中的模拟时钟立即推进下一事件的发生时间.如下一事件还未处理结束之前,另有其他事件等待处理,则这些事件就应依次排队等候处理.
模拟算法如下:

[算法]接待站模拟

```
{ 设置统计初值;
    设置当前时钟时间为 0;
    打开数据文件,准备读;
    读入第一位顾客信息于暂存变量中;
    do{          / *约定每轮循环,接待完一位顾客 * /
        if(等待队列为空,并且还有顾客){ / * 等待队列为空时 * /
            累计接待员总等待时间;
            时钟推进暂存变量中的顾客的到达时间;
            暂存变量中的顾客信息进队;
            读取下一位顾客信息暂存于变量;
        }
        累计顾客人数;
        从等待队列出队一位顾客;
        将该顾客的等待时间累计到顾客的总等待时间;
        设定当前顾客的接待结束时间;
        while(下一位顾客的到达时间在当前顾客接待结束之前){
                    暂存变量中的顾客信息进队;
                    读取下一位顾客信息暂存变量;
        }
```

```
        时钟推进当前顾客接待结束时间;
    }while(还有未处理的顾客);
    计算机统计结果,并输出;}
```

按上述算法编写的程序如下:

[例 8.4 程序]

```
#include<stdio.h>
#include<stdlib.h>
typedef struct { / * 顾客信息结构 * /
        int arrive;
        int treat;
} NODE;
typedef struct node { / * 队元类型 * /
        NODE data;
        struct node * link;
} LNODE;
LNODE * head, *tail;
NODE curr,temp;
char Fname[120];
FILE * fp;
/ * [函数]链接队列进队函数 * /
void len_queue(LNODE ** hpt, LNODE ** tpt, NODE x)
{ LNODE * p=(LNODE * ) malloc(sizeof(LNODE));
    p->data=x;
    p->link=NULL;
    if(*hpt==NULL) *tpt =* hpt=p;
    else
        * tpt= (* tpt)->link=p;
}

/*【函数】链接队列出队函数*/
int lde_queue(LNODE **hpt, LNODE ** tpt, NODE *cp)
{ LNODE * p=*hpt;
    if (*hpt= =NULL) return1; /*队空*/

    *cp=(*hpt)->data;
    *hpt=(hpt)->link;
    if (*hpt= =NULL) *tpt = NULL
    free(p);
    return 0;
}
void main()
{ int dwait =0 ,clock=0 , wait=0, count=0, have=0,finish;
```

```
    printf("Enter file name.")
    printf("%s",Fname)
    if (fp=fopen(Fname,"r"))= =NULL { /*打开文件*/
        printf("Can't open file %s", Fname);
    return;
}
head=NULL; tail=NULL;
have=fscanf (fp,"%d%d",&temp.arrive, &temp.treat); /*正确读入,have 为
2*/
do { /*约定每轮循环，接待一位顾客*/
    if (head= = NULL && have = = 2) { /*等待队列为空，但还有顾客*/
            dwait+=temp.arrive-clock; /*累计接待员总等待时间*/
            clock=temp.arrive; /*时钟推进到暂存顾客的到达时间*/
            len_queue(&head,&tail,temp); /*暂存变量中的顾客信息进队*/
            have=fscanf(fp,"%d%d",&temp.arrive,&temp.treat);
        }
    count++; /*累计顾客人数*/
    lde_queue(&head,&tail,&curr); /*出队一位顾客信息*/
    wait+=clock-curr.arrive; /*累计到顾客的总等待时间*/
    finish=clock+curr.treat; /*设定接待结束时间*/
    while (have= = 2 && temp.arrive<=finish){
    /*下一位顾客的到达时间在当前顾客接待结束之前*/
    len_queue(&head,&tail,temp); /*暂存变量中的顾客信息进队*/
    have=fscanf(fp,"%d%d", &temp.arrive, &temp.treat);
  }
  clock=finish; /*时钟推进到当前顾客接待结束*/
}while (have= = 2| |head! =NULL);
printf ("结果：接待员等待时间%d\n顾客平均等待时间%f\n",
        dwait, (double) walt / count) ;
Printf ("模拟总时间：%d，顾客人数；%d，总等待时间：%d\n',
        lock, count, walt) ;
}
```

8.4 数组

8. 4. 1 数组的基本概念

数组是最常用的数据结构之一，在程序中，数组常用来实现顺序存储的线性表。

在概念上，数组由固定个数的元素组成，全部元素的类型相同，元素依次顺序存储。每个元素对应一个下标，数组元素按数组名和元素的下标引用，引用数组元素的下标个数称为数组的维数。

在 C 语言中，约定数组的第 1 个元素的下标为 0，其余依次类推，由 n 个元素组成的数组，其最后一个元素的下标为 n－1。数组元素可以是任何类型的，当元素本身又是数组时，就构成多维数组。

数组通常是顺序存储的，对于多维数组，C 语言按行优先顺序存放。设有以下数组
定义：
int a1[10], a2[8][9], a3[7][8] [9] ;

则数组 a1 是一维数组，a2 是两维数组，a3 是三维数组.记元素 X 的内存地址为 D(x),设每个元素所需内存字节数为 s，存储元素 a1[i], a2[i][j], a3[i][j][k] 的内存地址分别可由以下算式确定:

D(a1[i])=D(a1[0])+I*s
D(a2[i][j])=D(a [0][0])+(I*9+j)*s
D(a3[i][j][k])=D(a [0][0][0])+((I*8+j)*9+k)*s

若用 C 语言的指针来描述，以上算式可分别简写成：

&a1[i]=&a1[0]+I
&a2[i][j]=&a[0][0]+I*9+j
&a3[i][j][k]=&a[0][0][0]+(I*8+j)*9+k

在使用数组的程序中，特别需要注意数组元素处理前和处理后的位置变化，以下例子说明这样的应用。

［例 8.5］ 设有 n 行 n 列两维数组，因其下三角元素全为零，不再存储其下三角的零元素，只将其上三角的元素按行优先存储于一维数组中。编制一个函数，要求函数在不引入工作数组情况下，将按行优先存储的上三角元素改为按列优先存储。

设 n=5，下三角元素全为 0 的两维数组 a 为

int a[ ][5]=
{1，2，3，4，5} ,
{0，6，7，8，9} ,
{0，0，10，11，12} ,
{0，0，0，13，14} ,
{0，0，0，0，15} } ;

不存储它的下三角 0 元素，按行优先存储于一维数组 b 中, b 为
int b[ ] = {1，2，3，4，5，6，7，8，9，10，11，12，13，14，15} ;

函数在不使用工作数组情况下，将使 b 的值成为：
{1，2，6，3，7，10，4，8，11，13，5，9，12，14，15} 。

函数是对数组元素的存储位置作调整，全部调整元素存在若干个循环传送链：
$b[i_0]\rightarrow b[i_1]\rightarrow \cdots \rightarrow b[i_0]$, 0<=j<n-1.

例如，考察调整后的 b［2］，与该元素调整相关的调整元素形成以下循环传送链：
b[2] →b［3］→b［6］→…→b［12］→b[9]→b［5］→b［2］。

实现上述调整的关键是确定循环传送链的下标链 i_0，i_1，…，i_j，i_0，以及在考察调整后的元素 b[k]时，能判定 b[k]是否是已被调整过的某传送链上的元素。

首先，对于给定的 n，因数组 b 的最前两个元素和最后两个元素的位置不变，要考虑调整的元素共有（n＋1）*n / 2-4 个。设函数自调整后下标 k=2 的元素开始顺序考察，考虑其调整前的下标 r。如 r 等于 k，或 b[k]是已被调整过的某传送链上的元素,b[k]就被掠过。否则，可这样调整 b[k]所在的传送链上的全部元素。令元素调整后的下标为 C（初值为 k），先将 b[k]保护到 w，然后在 r! = k 情况下循环完成以下工作：将 b[r]复制到 b［c］，再令元素调整后的下标 c 为 r，并由 c 求出它的调整前的下标 r。当循环因 r 等于 k 结束后，将 W 的值复制到 b［C］。至此，完成一个循环传送链上的全部元素的调整。

为判 b[k]是否是已被调整过的某传送链上的元素，如 b[k]是，则对当前的 k 来说，b[k]所在传送链中一定有一个元素的下标比 k 小。所以可用类似循环传送的办法来判定，将传送循环中的元素值复制工作去掉，并将循环条件改为 r 大于 k。当循环因 r 小于 k 而结束时，说明 b[k]是已被调整过的某传送链上的元素；当循环因 r 等于 k 而结束时，说明 b[k]是未被调整过的。

由以上分析发现，还需要一个已知元素调整后的下标 k，求其调整前下标的函数。该函数可以这样来考虑，由已知两维数组大小 n 和按列优先的上三角元素下标 k，设法求出其对应两维数组元素的行号 i 和列号 j。然后再由 i 和 j，求出按行优先的上三角元素下标。其中由 k 和 n 求 i 和 j 可利用以下事实：首先是 i 应不大于 j，其次是在上三角中 j 列元素共有 j+1 个。这样，可以令 i 的初值为 k，在 I>j 的条件下循环让 j 增 1，并从 i 减去 j，直至 I<=j。有了 i 和 j，利用 i 行有 n-i 个元素的事实，就能求出按行优先的下标。细节见下面的函数。

【函数】由按列优先存储下标为 k 的元素，求其按行优先存储的下标。

```
Int colToRow （Int k，Int n）
 {int i＝k，j=0;
     while（i＞j）i-=++j;
     return I*n+ j-I*（i＋1）/2;
}
```

【函数】上三角阵一维数组按行优先存储改为按列优先存储

```
void trans（Int b［］，int n）
 { int m, k, c, w ;
```

```
      m=(n+1)*n/2-4;
      k=2;
      while (m>0) { /*每调整一个减 1*/
      r=colToRow (k,n); /*调整后下标为 k 的元素，求得调整前的下标 r */
      if (r= = k) m--; /*调整前后相同，不调整*/
      else{ /*调整前和后不相同，存在一个调整链*/
           c=k; w=r;
           while (r>k) /*遍历循环传送链，看循环结束前是否有比 k 更小的
下标*/
           c=r; r=colToRow(c,n);
           }
      if (r = = k ) { /*循环传送链上没有比 k 更小的下标，是还未处理的
传送链*/
           c=k; r=w; w=b[k];
           while (r !=k) /* 直至循环传送链又回到头，才结束*/
           b[c]=b[r]; m--;
           c=r; r=colToRow(c,n);
           }
           b[c]=w; m--;
      }
     }
     k++; /*准备考察下一个下标*/
   }
}
```

［例 8．6］编制利用辗转相除法求两个多项式的最大公因式的函数。

设多项式的系数按幂次由高到底的顺序存于一数组中，多项式的幂次存于另一变量中。辗转相除法求两多项式的最大公因式的算法是：用其中的一个多项式去除另一个多项式，然后将所得余式变成除式，原除式变成被除式。如此反复相除，直至余式为零时，最后的除式即为最大公因式。细节见以下函数。

【函数】用辗转相除法求两多项式最大公因式

```
Int Remalnder（double*pA，Int an，double*pB，Int bn，double**q）
/*已知两多项式的系数和幂次，返回最大公因式的幂次，在*q 得到系数表的首
指针*/
{   double x，y，*temp ;
      int k，J，flg=1;
       if（an＜bn）{/*使多项式 A（x）的幂次不低于多项式 B（x）的幂次
*/
          temp＝pA; pA＝pB; pB＝temp;
          k＝an; an＝bn; bn＝k;
      }
    while（flg） {/*循环至余式为 0 结束*/
```

```
          while（*nB= = 0. 0 ＆＆bn ＞ 0）｛／*掠过多项 B（x）最高次系数为 0 的项／
               bn--; PB++;
          }
     k＝0; X=*pB;
     while（k ＜= bn pB[k++]／= x /*使多项式最高次项的系数归约为 1*/
     for（k＝0; k ＜＝an-bn; k＋＋ ｛ ／*A（x）／B（x），商多项式有 an-bn＋1 项*／
          x=pA[k]; ／*商多项式的当前项系数 X=pA[k]*／
          for ( j =0, j<bn, j++) ｛ ／*从 A（X）中减去 X*B（X）*／
          y=pA[k+j+1]-x*pB[j+1]; ／*pAk 应置 0，因不再使用，该值未置 0*／
          pA[k+j+1]=(y<.00005 && - y < .00005)? 0.0 : y;
          }
     }
     temp=pA; pA=pB; pB=temp+an-bn+1; an=bn--;
     for ( flg=0,k=0; k<=bn && flg= = 0; k++) ／*判多项式 B(x)是否为 0 */
          if（pB[k]!=0.0） flg=1;
     }
     *q=pA;
     return an;
}
```

[例 8. 7] 几个长整数运算函数。

在程序设计中，常遇长整数处理问题。上百位的长整数可简单地用数组来存储，数组的每个元素顺序存储长整数的一位数字。设有 k（<200）位长整数 m，用数组 a [] 存储：

$$m=a[k]*10^{k-1}+a[k-1]*10^{k-2}+\cdots+a[2]*10^{1}+a[1]*10^{0}$$

并用 a[0]存储长整数 m 的位数，即 a[0]=k。以下是一些长整数处理中常用的函数。

【函数】输人长整数函数

输人长整数函数 readMaxlnt（int*a）将长整数存于数组 a，并返回该长整数的位数。

```
# define MAXN 400
Int readMaxlnt（Int*a）
 {     int len, i;
        char s[MAXN];
         printf("Inter long integer:\n");
          scanf ("%s",s);
```

```
          len=strlen(s);
          for (I = len; I>=1, I--)
               a[i]=s[len-i]-'0';
          return a[0]=len;
}
```

【函数】输出长整数函数

```
void writeMaxint (Int*a)
{ int I ;
     for (I=*a; I>=1; I--)
     printf("%d",a[i]);
     printf("\n");
}
```

【函数】长整数规整函数

如上所述，存储长整数数组的每个元素只存储长整数的一位数字。但在长整数运算过程中，会使存储中间结果的长整数数组元素自值大于 9，这时，就要用规整函数将存储它的数组的每个元素只存储长整数的一位数字。

```
Int formal (Int * a)
{ int p;
     for (p=1; p<a[0] ‖a[p]>=10; p++) {
     if (p>=a[0]) a[p+1]=0;
     a[p+1]+=a[p]/10;
     a[p]=a[p]%10;
}
if (p>a[0]) a[0]=p;
     return a[0];
}
```

【函数】长整数除普通整数函数

长整数除普通整数运算函数 divInt（int * a，int j）实现将存于 a 中的长整数除普通整数 j，并将商存于 a 中，函数返回整除后的余数。

```
Int divInt(int *a,int j)
{ int p, k;
     k=p=a[0]; a[0]=0 ;
     while ( p>0 ) {
     a[p-1]+=a[p] % j * 10;
     a[p]=a[p]/j ;
     if ( a[k] = = 0) k --;
          p --;
```

```
    }
    p = a[0]
    a[0]=k ;
    formal (a);
    return p ;
}
```

【函数】将长整数转换成二进制长整数函数

将十进制长整数转换成 M 进制长整数函数 dec2bin（int *a，int *b），该函数将存于 a 中的长整数转换为二进制，并存于整数数组 b 中。以下函数采用除 2 取余法转换，即用 2 不断去除待转换的十进制数，直至商为 0 为止。将所得的各次余数依次排列存储，并以最后余数为最高位，即得到转换后的二进制数的长整数表示。

```
void dec2bin（int *a，int *b）
{  int p;
    b[0]=0;
    while ( a[0]>0) {  /*用 2 不断去除待转换的十进制数，直至商为 0 为*/
        b[0]++ ;
        b[b[0]] = a[1] % 2;
        p = a[0];
        while (p>0) {  /*对 a 执行除 2 运算*/
            if ( a[p] % 2 && p>1) a[p-1]+=10;
            a [p]/=2 ;
            if ( a[a[0]] = = 0 ) a[0] -- ;
            p -- ;
        }
    }
}
```

【函数】计算普通整数的组合函数

两个普通整数的组合数可能是一个大整数，为此两整数的组合用长整数存储。设两个普通整数是 a 和 b，并且有 a＞b。它们的组合 c（a，b）= a！/（a-b）！b！）。计算 a 和 b 的组合可采用以下方法：

设 u＝u1u2…ub，d＝d1 d2…db。

其中 u1=a，u2=a-1，…，ub=a-b＋1

d1＝1，d2＝2，…，db＝b

显然 u1u2…ub=a(a-1)(a-2)…（a-b＋i）=a！/（a-b）！

a 和 b 的组合 c（a，b）为

（u1u2…ub）/（d1 d2…db）

为计算上述分数，可先从 u1u2…ub 中去掉所有 d1 d2…db 的因子，得到新的 u1u2…ub.然后再将它们相乘。其中去因子可采用以下代码实现（以下代码调用的函数 gcd 是求两整数最大公因子函数）：

```
for (I=1; I<=b; I++)
     if (u[i]!=1)
          for (j=1; j<=b; j++)
               if (d[j]!=1) {
                    x= gcd (u[i], d[j]);
                    u[i]=u[i]/x; d[j]=d[j]/x;
```

而相乘可采用以下代码:

```
c[0]=1;c[c[0]]=1;
for(I=1;I<=b;I++)
if (u[i]!=1){/ * 对每个 u[i]乘长整数的各位 * /
for(j=1,j<=n;j++)c[j]=c[j ]* u[i];
/ * 对长整数进行规整; * /
}
```

最后得到以下计算精通整数组合的函数:

```
void combine(int a,int b,int * c)
{ int i, j ,x,k;
ind d[MAXN],u[MAXN];
for(k=0,I=a;I>=a-b+1;I--)u[++k]=I;
u[0]=k;
for(i=1;I<=b;I++)d[i]=I;
for(I=I;I<=u[o];I++)
if(u[i]!=1)
for(j=1;j<=b;j++)
if(d[j]!=10&u[i]!=1){
x=gcd(u[i],d[j];
u[i]/=x;d[j]/=x;
}
for(c[0]=c[1]=I=1;I<=u[0];I++)
if(u[i]!=1){
for(j=1;j<=c[0];j++)
c[j]=c[j] * u[i];
formal(c);
```

```
}
}
```

[函数]求两个整数和的函数

设求两长整数和的函数 add(int * a,int * b)计算存于 a 与 b 中的两长整数之和存于数组 a.

```
Void add(int * a,int * b)
{ int I;
     while (a[0]<b[0]) / * 如 b 的位数多,在 a 的首部添 0 * /
     a[++a[0]]=0;
     for(I=1;I<b[0];I++)
     a[i]+=b[i];
     formal(a);
}
```

8.4.2 稀疏矩阵

在一个矩阵中,如其零元素的个数大大多于其非零元素的个数时,称这样的矩阵为稀疏矩阵.若直接用一个两维数组表示稀疏矩阵,会因存储太多的零元素要浪费大量的内存空间.通常采用三元数组或十字链表两种方法来存储稀疏矩阵.

1. 用三元组数组表示稀疏矩阵

稀疏矩阵的每个非零元素用一个三元组来表示,即非零元素的行号、列号和它的值。然后按某种顺序将全部非零元素的三元组存于一个数组中。如对于以下两维数组

```
int x [5][4]={{1,0,0,0}
                                {0,5,0,0}
                                {0,0,7,2}
                                {6,0,0,0}
                                {0,3,0,8}};
```

可用以下数组 a 表示：

```
int a[][3]={{5,4,7},
                            {0,0,1},
                            {1,1,5},
                            {2,2,7},
                            {2,3,2},
                            {3,0,6},
                            {4,1,3},
                            {4,3,8}};
```

其中三元组数组 a 的第一行元素的值分别存储稀疏矩阵 x 的行数、列数和非零元素个数。

[例 8.8]将稀疏矩阵 a 转置生成稀疏矩阵 b。设稀疏矩阵的非零元素已按行号递增，同一行按列号递增的顺序以三元组形式存于某数组中，要求对稀疏矩阵作转置操作，并使转置后的元素依然与原来相同的按行优先顺序存储形式存放。

实现上述要求有多种方法可以选用，这里的函数采用以下算法实现将稀疏矩阵 a 转置生成稀疏矩阵 b:

[算法]生成用三元组数组表示的稀疏矩阵。

{置 b 的三元组数组的第 1 行的值；

确定 a 各列元素个数（即 b 的各行元素个数）；

求出 b 的各行第一元素的存储下标；

顺序访问 a 的各元素

{将 a 的当前元素按其列号复制到 b 的对应行中；

修正该行下一个元素的存储下标；
   }
}

[函数]

```
void transpose(int a[][3],int b[][3])
{  int m,n,I,j,* x,*,y;
    b[0][1]=a[0][0]; / * 转置后矩阵的列数 * /
    n=b[0][0]=a[0][1]; / * 转置后矩阵的行数 * /
    m=b[0][2]=a[0][2]; / * 转置后矩阵的元素个数 * /
    if(m<=0)return;
    y=(int * )malloc(sizeof(int) * (n+1)); / * 工作数组 * /
    x=y+1;
    for(I=0;I<n;I++)x[i]=0;
    for(I=0;I<=m;I++)x[a[i][1]]++; / * 统计 b 的各行元素个数 * /
    y[0]=1;
     for(I=1;I<n;I++)
    y[i]=y[I-1]+x[I-1]; / * 计算 b 的各行第一个元素的存储下标 * /
    for(I=1;I<=m;I++){
    j=y[a[i][1]]++; / * 增 1 是为了预置同行下一元素的存储下标 * /
    b[j][0]=a[i][1] / * 列号改为行号 * /
    b[j][1]=a[i][0]; / * 行号改为列号 * /
```

```
    b[j][2]=a[i][2]; / * 复制值 * /
    }
    free(y);
}
```

2. 用十字链表示稀疏矩阵

如果对稀疏矩阵某行或某列整体作某种处理，可能会使原来为零的元素变为非零，而原来非零的元素可能变成零。对于这种场合，稀疏矩阵宜用十字链表来表示。在十字链表中，矩阵的非零元素是一个终点，同一行的结点和同一列的结点分别顺序环形链接，每个结点既在它所在行的环形链表中，又在它所在列的环形链表中。每个结点含 5 个字段：结点对应的矩阵元素的行号、列号、值，以及该结点所在行链表后继结点指针、所在列链表继结点指针。结点类型可定义如下：

```
typedef struct ele { / * 十字链表结点类型 * /
            int row,col;
            double val;
            struct ele * right,* down;
}eleNode;
```

通常为处理方便，对每个行链表和列链表分别设置一个表头结点，并使它们构成带表头结点的环形链表。为了引用某行某列的方便，全部行链表的表结点和全部列链表的表头结点又把它们分别组成数组，这两个数组的首结点指针存于一个十字链表的头结点中，最后由一个指针指向该头结点。细节见以下十字链表表示稀疏矩阵的几个常用操作函数。

[函数]建立一个元素全为零的稀疏矩阵的十字链表。

```
EleNode * createNulMat(int m,int n) / * m 行 n 列 * /
{ eleNode * h, * p;
   int k;
   h=(eleNode *)nalloc(sizeof(eleNode)); / * 十字链表头结点 * /
   h->row=m;h->col=n;h->val=0; / * 稀疏矩阵的行数和列数 * /
   h->right=(eleNode )malloc(sizeof(eleNode) * n);
   h->down=(eleNode)malloc(sizeof(eleNode)* m);
   for(p=h->down,k=0;k<m;k++,p++){
      p->col=1000; / * 设矩阵不会超过 1000 列 * /
      p->right=p; / * 每个行链表是一个环 */
      p->down=k<m-1?p+1:h->right; /* 使全部列链表头结点构成环 * /
   }
   for(p=h->right,k=0;k<m;k++,p++){
      p->row=1000;
      p->down=p;
      p->right=k<m-1?P+1:h->rigth;
   }
```

```
return h;
}
```

[函数]在十字链表中插入一个结点

```
int inserNode(eleNode * a,int row, int col,double val)
{   eleNode * p,* q ,* r, * u,* v;
      if(row>=a->row||col>=a->col)return-2; /* 不合理的行列号* /
      r=(eleNode * )malloc(sizeof(eleNode));
      r->row=row;r->col=col;r->val=val;
      p=a->down+row;q=p->right;
      while(q->col<col){p=q;q=q->right;}
      if(q->col==col){free}(r);return-1;} / * 该行已有 col 列元素 * /
      u=a->right+col;v=u->down;
      while(v-row<row){u=v;v=v->down;}
      if(v->row==row){free(r);return-1;} / * 该列已有 row 行元素 * /
      p->right=r;r->right=q; / * 插入行链中 * /
      u->down=r;r->down=v; / * 插入列链中 * /
      a->val+=1.0;
      return 0; / * 插入成功 * /
}
```

[函数]输入数据建立十字链表。

设稀疏矩阵的输入数据按以下格式要求输入：

·首行输入稀疏矩阵的行数和列数。

·若某行有非零元素，则按如下格式输入：
行号 列号 值 列号 …列号 值 -1
最后的-1 表示该行的一组数据输入结束。

·按以上格式一行行地输入数据，直到新输入的行号的值为-1 时，表示稀疏矩阵的全部数据输入结束。

```
EleNode * readMat()
{ eleNode * h;
      int I,j,m,n;
      double v;
      printf("输入稀疏矩阵的行数和列数");
      scanf("%d%d",&m,&n);
      h=createNullMat(m,n);
      h->row=m;h->col=n;
      printf("输入有非零元素的行号");
      scanf("%d",&j);
```

```
      while(j>=0){ / * 输入一列非零元素 * /
      printf("输入非零无素的值");
      scanf("%lf",&v);
      printf("输入当前行下一个非零元素的列号（-1 表示当前行结束）");
      scanf('%d",&j);
}
      printf("输入下一列有非零元素的行号（-1 表示输入结束）");
      scanf("%d",&i);
   }
   return h;
}
```

[函数]两个相同行数和列数的稀疏矩阵加法运算。

```
EleNode * matAdd(eleNode * a,eleNode * b)
{   eleNote * r,* p,* q,* u,* v;
      r=createNullMat(a->row,a->col);
      p=a->down;u=b->down;
      while(p!=a->down){ / * 逐行相加 * /
         q=p->right;v=u->right;
         while(q!=p||v!=u) / * 两矩阵有一个行未结束循环 * /
            if(q->col==v->col){ / * 有相同列的元素 * /
               if(q->val+v->val!==0.0 ) / * 和非零插入 * /
                     insertNode(r,q->row,q->col,q->val+v->val);
                  q=q->right;v=v->right;
            }
      else if(q->col<v->col){
            insertNode(r,q->row,q->col,q->val);
            q=q->right;
      }
         else{ / * 插入 b 的元素 * /
            insertNode(r,v->row,v->col,v->val);
             v=v->right;
            }
         p=p->down;u=u->down; / * 或 p++;u++ * /
      }
      return r;
}
```

8.5 字符串

字符串已成为非数值处理应用重要的处理对象。如文字编辑、情报检索、自然语言翻译和各种事务处理等。

字符串是由某字符集上的字符所组砀任何有限字符序列。当一个字符串不包含任何字符时，称它为空字符串。一个字符串所包含的有效字符个数称为这个字符串的长度。一个字符串中任一连续的子序列称为该字符串的子串。

在 C 语言中，字符串常量是用一对双引号括住若干字符来表示，如下面的例子：
″ I am a string″
″″ / * 一个空字符串 * /

字符串通常存于足够大的字符数组中。另外，每个字符串的最后一个有效字符之后有一个字符串结束符，记为'\0'。通常由系统提供的库函数形成的字符串的末尾会自动添加字符串结束符，但当由用户的应用程序来形成字符串时，必须由程序自行负责在列之后添加字符串结束符，以形成字符串。

C 系统为字符串的各种运算提供了高效、便于使用的库函数，这里给出若干实例，说明字符串处理中的一般方法。

[例 8. 9]编制一个函数，实现在两个已知字符串中找出所非空最长公共子串的最长度和最长公共子串的个数。

设两个字符串首字符指针分别为 str1 和 str2，它们的长度分别记为 len1 和 len2,不失一般性,设有 len1≥len2。则它们最长的公共子串长度不会超过 len2。程序为找最长的公共子串，考虑找指定长度的所有公共子串的子问题。当指定长度从 len2 开始逐一递减的寻找过程中，一旦找到了公共子串的话，程序最先找到的公共字符串就是最长公共子串。按上述想法编写的程如下：

[例 8. 9 程序]

```
#include < stdio.h>
#include <string.h>
int commStr(char * strl,char * str2,int * lenpt)
/* 函数返回最长公共子串的个数,通过指针参数传回最长公共子串的长度过*/
{ int len1,len2,in, count ,I,k,p;
      char * st
      if((len1=strlen(strl))<len2=strlen(str2))){
      st=str1;strl=str2;str2=st;
      ln=len1;len1=len2;len2=ln;
   }
   count=0;
   for(in=len2;in>0;in--){
            / * 找长为 ln 的公共子串 * /
      for(k=0;k+ln<=len2;k++){
            / * 自 str2[k]开始的长为 ln 的子串与 strl 中的子串比较 * /
         for(p=0;p+ln<=len1;p++){
            / * strl 中的子串自 str[p]开始，两子串比较通过对应字符逐一比较实现 * /
```

```
            for(I=0;I<ln;I++)
               if(str2[k+i]!=str1[p+1])break;
            if(I==ln) / * 找到一个最长公共子串 * /
               count ++;
         }
      }
      if(count)break;
   }
   *lenpt=ln;
   return count;
}
void main()
{   int c,len;
      c=commStr("AbclAbcsAbcd123","123bAbc",& len);
      printf("有%d 个长为%d 的公共子串\n",c,len);
}
```

[例 8.10]编制一个字符整理函数 ver(char * s)，实现将已知字符串 S 中的前导空白类字符和尾随的空白类字符去掉，并将字符串非空白类字符之间的连续多个空白类字符缩减为一个空白符。

函数引入一个用于重新生成字符串 S 的指针变量，首先用一个掠过前导空白类字符的循环，去掉字符串 S 的其余字符，对每个顺序访问的当前字符，检查该字符是否是空白类字符，若其最后一个复制字符为空白符，则把该空白符改成字符串结束符号，如此处理就能保证字符串 S 中的尾随空白符全部去掉。写成函数如下：

```
[函数]
char * ver(char * s)
{   char * p=s, *q=s;
      / * 掠过前导空白符 * /
      for (; * s==''|| * s= = '\t' || * s= ='\n'; s++);
      for(; * s ;){ / * 顺序访问字符串 S 中的每个字符 * /
         *q++ = (*s= ='\t'|| * s= ='\n') ?'' :* s;
         if( * s!='' && * s!='\t'&& * s!='\n')s++;
         else / * 掠过中间连续的多余空白符 * /
               while (* s= =''|| * s= ='\t'||* s= ='\n')s++;
         }
         if(q>p && * (q-1)= ='') / * 如 q>p，则已复制过非空白符 * /
            *（q-1）='\0'; / * 如最后复制的是空白符，将其改为字符串结束符 * /
         else
            * q='\0'; / * 否则添加字符串结束符 * /
         return p;
}
```

对于本题，也可这样设计求解方法，函数实际上也是要完成一个字符串的拷贝，但对其中空白类字符串 S 中的位置不同有不同的处理要求。通常对同一数据对象，因其所在位置不同有不同的处理要求。对于类似情况，习惯引入一个称为状态变量的整型变量，开始时让 flg 为 0，表示程序将遇到的空白类字符是字符串的前导空白类字符；flg 为 1，表示在字符复制过程中遇到了一个中间空白类字符列；flg 为 2，表示一个空白类字符列已处理结束。在顺序访问字符串的字符过程中，遇空白类字符时就掠过它不拷贝，但在 flg 为 2 的情况下，遇到一个空白类字符的其他字符时，若 flg 为 1，则在复制当前字符之前，插入复制一个空白符，接着再复制字符和置 flg 为 2，表示一个中间空白字符理处理结束。写成函数如下；

[函数]

```
char * ver(char * s)
{   int flg=0;
      char * p=s, * q =s;
      for(; * s;s++){ / * 顺序访问字符串 s 中的每个字符号* /
      if( * s==''|| * s=='\t'|| * s== '\n') / * 遇空白类字符 * /
         flg=(flg==2)?1: flg;
      else{ / * 当前字符是其他字符
         if(flg==1)
            *q++=' '; / * 遇到过空白类字符，复制一个空白符 * /
            *q++=*s;flg=2; / * 一个中间空白符列处理结束语* /
         }
      }
      * q='\0'
   return p;
}
```

接受各种可能的字符序列，作某种处理的程序可编写成一种识别程序。将程序可能接受的字符按其作用分类，为记录程序的识别过程引入状态变量，并给所有可能的状态顺序编号。然后建立两个表，一是状态转换表，反映在某状态下遇某类字符后程序应执行的识别语义动作，以本例问题为例说明这种方法的用法。首先函数将可能接受的字符可分成三类:空白类字符、非空白类字符和字符串结束符。引入状态变量 status 表示识别程序所处状态。令 status 为 0，表示函数还在处理字符列的前导空白类字符：status 为 1 表示函数处理非空白类字符号后的空白类字符；status 为 2 表示正在处理非空白类字符；最后，status 为 3 表示函数已遇到了字符串结束符，识别过程应结束。其中状态转换表和语义操作表分别见程序中的数组 stable[][]和 funtbl[][].

[例 8. 10 程序]

```
# include<stdio.h>
#include<string.h>
void copyach(char * * q,char ch) / * 复制一个非空白类字符 * /
```

```
{ *((*q)++)=ch
}
void copybach(char * * q,char ch) / * 复制一个空白符和非空白符 * /
{    * ( (* q) ++) =' ';
      *((*q)++)=ch;
}
/ * 状态转换表* /
int statbl[][3]={{0,2,3}, / * 初始状态 * /
                                  {1,2,3}, / * 中间空白符状态 * /
                                  {1,2,3}, / * 非空白符状态 * /
/ * 语义操作表 * /
void(* funtbl[][3](char * * ,char)={NULL,copyach,NULL},

{NULL copybach ,NULL},

 {NULL copyach,NULL};
char * ver (char * s) / * 识别函数 * /
{int c,ckind,
   status=0; /* 置初始状态 * /
   char * p=s,*q=s;
   for(;status!=3;s++){ / * 顺序访问字符串 s 中的每个字符 * /
         c=* s;
               / * 分类当前字符 * /
         if (c= = ' '||c= ='\t'||c= ='\n')
               ckind=0; / *空白类字符 * /
         else if (c= ='\0') ckind=2; / * 字符串结束符 * /
               else ckind=1; / 非空白类字符号 * /
         if(funtbl[status][ckind]) / * 如有转换函数 * /
            (* funtbl[status][ckind])(&q,c); / * 执行相应函数 * /
         status=stabl[status][ckind]; / * 设置新的状态 * /
      }
       *q='\0';
      return p;
}
void main()
{   char sp[120],*ss;
      printf("输入字符串.\n");
      gets(s);
      ss=ver(s);
      printf( "结果字符串为%s,其长度为%d",ss,strlee(ss));
}
```

[例 8.11]用已知字符串 s 中的字符,生成由其中 n 个字符组成的所有字符排列.设 n 小于字符串 s 的字符个数,其中 s 中的字符在每个排列中最多出现一次.例如,对于

s[]="abc",n=2

则所有字符排列为:

ba,ca,ab,cb,ac,bc.

采用递归方法,取字符串 s 中的字符生成由 n 个字符组成的字符排列.设函数用字符组 W[]存储生成的字符排列.令递归函数为 perm(int n, char * s),其功能是用字符串 s 中的字符,生成由 n 个字符组成的所有排列.其方法是从字符串 s 依次选用其中的每个字符填写到存储字符排列数组 W[]的第 n 个位置(w[n-1])中,然后通过递归调用生成由 n-1 个字符组成的排列.当一个字符被选用后,进一步递归调用时可选用的字符中应包含该字符.另外,当发现一个字符排列生成后,就不再递归调用,而是输出生成的字符列.综合以上分析,函数 perm(int n,char * s)的算法如下:

[算法]函数 perm(int n,char * s)的算法.

```
{   if(一个排列已生成)printf("% s",w); / * 一个排列生成输出 * /
      else{
          保存本层次可使用的字符串;
          依次选本层次可用字符循环完成以下工作{
             将选用字符填入正在生成的字符排列中;
             形成进一步递归可使用的字符串;
             递归调用;
          }
      }
}
```

由以上分析和算法可知,当递归调用生成 0 个字符的排列时,一个字符排列已生成.为保存本层次可使用的字符串,可引入一个字符数组.为存储进一步递归可使用的字符串,可作以下技术处理,因进一步递归可使用的字符总比本层次可使用的字符少一个字符,可把本层次正选用的字符与其字符串中的首字符交换,而进一步递归可使用字符串就是从第二个字符开始的字符串.

[例 8.11 程序]

```
#include<stdio.h>
#include<string.h>
#define N 20
char w[N];
void perm(int n,char * s)
```

```
{   char s1[N];
      int I;
      if(n<1)printf("%s\n",w); / * 一个排列生成输出 * /
      else { / * 设定本层次字符,并递归调用生成更深层次字符 * /
            strcpy(sl,s); / * 保存本层次可使用的字符 * /
            for(I=0;* (sl+I);I++){ / * 依次选本层次可用字符 * /
               *(W+n-1)=* (sl+I); / * 选用字符填入正生成的字符排列中 *
/
               *(sl+i)=* sl;
               sl =*(w+n-1);
               perm(n-1,sl+1); / * 递归 * /
      }
   }
}
void main()
{ perm(2,"abc");
}
```

[例 8.12]KMP 模式匹配算法

在长正文字符串 t 中,找与模式字符串 p 相同子串的过程称为模式匹配.最简单直观的模式匹配算法的核心工作是确定从字符 t 第 j 个字符开始的子串是否能与字符串 p 相同.当 j 从 0 开始依次递增变化时,如对于某个 j 找到从 t[j]开始的子串与字符串 p 相同,就是匹配成功,结束匹配过程.如直至扫视了整个 t,都找不到与字符串 p 相同的子串,就说匹配失败.

按该思想写成函数如下:

[函数] 简单模式匹配算法。

```
int simple-match(char * t,char * p)
{   int n=strlen(t) ,m =strlen(p),I,j,k;
      for(j=0;j<n-m;j++){ / * 顺序考察从 t[j ]开始的子串 * /
            / * 从 t[j ]开始的子串与字符串 p 比较 * /
            for(I=0;I<m &&t[j+i]==p[i];I++);
            if(I==m)return 1; / * 匹配成功 * /
      }
   return 0; / * 匹配失败 * /
}
```

一个充分利用模式字串性质,能大大加快模式匹配过程的算法是 KMP(Knuth-Morris-Pratt)算法.在执行简单匹配算法中的字符比较过程中,一旦出现 t[j+i]!=p[i]时,有

t[0] …… t[j] t[j+1] … t[j+i-1] t[j+i] …

```
                ‖       ‖              ‖       !=
              p[0]    p[1]           p[i-1]  p[i]
```

这时,简单匹配算法对字符串 t 要回溯,回溯到从 t[j+1]开始继续与模式字符串 p 从头开始逐一字符比较,KMP 算法是对正文字符串比较不回溯的算法.

如果对于模式字符串 P 存在一个整数 K(K<I),使得模式中 P 开头的 K 个字符 (p[0],p[],…,p[k-1])依次与 p[i] 的前面 k 个字符 (p[I-k],p[I-k-1],…,p[I-1])相同，即

```
        p[0]  …… p[i-k]  p[i-k+1]   …    p[i-1]   p[i]    …
                   ‖        ‖              ‖       ?
                 p[0]     p[1]     …    p[k-1]   p[k]    …
```

由以上两个对应关系，就有

```
t[0] …  t[j]  …  t[j+i-k]  …  t[j+i-1] t[j+i] t[i+1]  …
          ‖          ‖            ‖       !=
        p[0]  …   p[i-k]   …   p[i-1]   p[i]    …
                    ‖             ‖       ?
                  p[0]    …   p[k-1]   p[k]    …
```

因此，可以从模式 p 的 p[k]开始从正文 t 的 t[j+i]开始依次继续比较，这就省去前面的 k 次比较。如果对应 p[i]的 k 有多个，应取最大的 k。

这样，在匹配比较过程中，一旦出现 p[i]！=t[j]时，就要找出 p[i]的 k，称 k 为 p[i]的失败链接值，寻找 p 的各字符的失败链接值，只与模式 p 本身相关，与正文 t 无关。求 p 的各字符的失败链接值算法如下：

设置一个一维数组 next[]，其元素个数与 p 的长度相同，依次存放 p 的各字符的失败链接值。首先置 next[0]=-1,然后假定在 next[0],…,next[j-1]已知的情况下,求 next[j].如果 next[j-1]=k,又有 p[k]=p[j-1],那么置 next[j]为 next[j-1]+1；如果 p[k]!=p[j-1],令 k1=next[k],如有 p[k1]=p[j-1],则置 next[j]为 k1+1 ；如果 p[k]!=p[j-1],则按 p[k1]的失败链接值再继续寻找，直至找到一个失败链接值 kn，使得 p[kn]=p[j-1],或者 kn=-1，这时都置 next[j]=kn+1.

以下分别是寻找模式各字符失败链接值的函数 failink（）和 KMP 模式匹配函数 kmp-match().

[函数]寻找模式各字符失败链接值的函数

```
void failink(char * p,int next[])
{   int j,k ;
      next[0]=-1;j=1;
```

```
      do {
         k=next[j-1];
         while (k!=-1&&p[k]!=p[j-1])
               k=next[k];
         next[j++]=k+1;
   }while(p[j]!)='\0');
}
```

[函数]KMP 模式匹配函数

```
int kmp_match(char * t,char * p,int next[])
{   int k,j;
      if(* t=='\0')return 0;
      for (k=j=0;t[k]!='\0';)
         while (j!=-1&&p[j]!=t[k])
               j=next[j];
         k++;j+=;
      if(p[j]=='\0')
            return k-j;
   }
   return -1;
}
```

8.6 树和二叉树

树和二叉树是非线性数据结构,能用它们很好描述有分支和层次特性数据集合.习惯称树和二叉数据为树形结构,树型结构在现实世界中广泛存在,如各种社会组织机构的组织关系图就可用树形结构来表示.树在计算机领域中也有广泛的应用,如在编译系统中,用树表示源程序的语法结构,在数据库系统中,树形结构是数据库层次模型的基础,也是各种索引和目录的主要组织形式.在许多算法中,常用树形结构描述问题的求解过程或表示求解的对策等.

由于二叉树有确定的分支个数,又可以为空,有非常良好的递归性质,与一般树相比较,特别适宜于程序设计,因此也常将树转换为二叉树进行处理.

8.6.1 树

1. 树的基本概念

树是一种多层支多层次数据结构,由一组结点组成,由于它呈现与自然界的树类似的结构形式,所以称它为树.
树的定义:树是由一个或多个结点组成的有限集 T,它满足以下两个条件:

(1) 有一个特定的结点:称为根结点;

(2) 其余的结点分成 m(m≥0)个互不相交的有限集.其中每个集合又都是一棵树,称. T1,T1,…,Tm-1 为根结点的子树.
以上定义是递归的,即一棵树由子树造成,子树又由更小的子树构成.由条件(1)知,一棵树至少有一个结点.在树中,那个惟一的特定结点没有前驱.一个结点的子树数目,称为结点的度(也可为结点的次数),树中各结点的度的最大值称为树的度(或树的次数),度为 0 的结点.称为叶子结点,除叶结点外的所有结点又称为分枝结点,根以外的分枝称为内部结点.如图 8.3 所示的树,根结点的度数为 3,结点 2 和结点 7 的度数为 2,结点 4 的度数为 1,该树的度数 3.

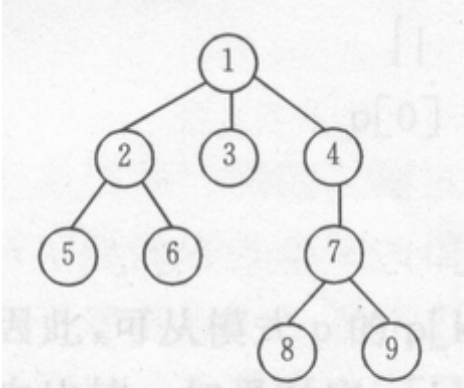


图 8.3 一棵树

在用图形表示的树中,对两个用线段连接的相关联的结点,称位于上端的结点是位于下端的结点的父结点或双亲结点;称位于下端的结点是位于上端的结点的子结点;称同一个父结点的多个子结点为兄弟结点.如结点 1 是结点 2、结点 3 和结点 4 的父结点。反之，结点 2、结点 3 和
结点 4 都是结点 1 的子结点。结点 2、结点 3 和结点 4 是兄弟结点。

定义一棵树的根结点所在的层次为 0，其他结点所在的层次等于它的父结点所在的层次加 1。对于树中两个不同的结点，如果从其中一个结点出发，自上而下沿着树中连着结点的线段能达到另一结点，称它们之间存在从一个结点到另一个结点的一条路径。可用路径所经过的结点序列表示路径，路径的长度等于路径上的结点个数和减 1。如图 8.3 中，结点 1 到结点 8 存在一条路径，并可用(1.4.7.8)表示这条路径的长度为 3。位于两棵不同子树上的结点之间不存在路径，如结点 2 与结点 9 之间不存在路径。特别地，从树的根结点出发，到树中的其余结点一定存在一条路径。

一般地，当树中某结点有多棵子树时，其子树的顺序是任意的。如果对一棵度为 m 的树,给树中的每个结点的每棵子树都规定了它们的序号,那么这棵树是一棵有序树.约定从左到右用整数 0,1,…m,-1 给结点的子树规定序号.

2.树的常用存储结构

因为树是非线性的结构,所以一般不能简单地用结点和线性表来表示.为存储树,必须要把树中结点之间存在的关系反映在存储结构中.树有多种实用的存储结构,这里只介绍其中最常用的树的标准存储和带逆存储形式。

（1） 树的标准存储结构

在树的标准存储结构中，树中的结点内容可分成两部分：

结点的数据 指向子结点的指针数组 对于 M 度树，在其标准存储结构中指针数组 M 个元素。另外要有一个指针变量，它指向树的根结点。

设树的次数为 10，树的结点数据仅限于字符，用 C 代码描述树结点的标准存储结构的数据类型如下：

```
#define M 10
typedef struct tnode{
     char data; /*树结点的数据信息*/
     struct tnode * child [M]; /*树结点的子结点指针*/
}TNODE; /*树结点的数据类型*/
TNODE * root ; /*树的根结点指针*/
```

(2)树的带逆存储结构

当程序需从结点返回到其父结点时，需要在树的结点中存有其父结点的位置信息，树的这种存储形式就是带逆存储结构。即在树的标准存储形式基础上增加一个指向其父结点的指针，用 C 代码描述树结点的带逆存储结构的数据类型如下：

```
#define M 10
typedef struct rtnode{
     char data; /*树的结点数据信息*/
     struct rtnode child[M]; /*树结点的子结点指针*/
     struct rtnode *father; /*父结点指针*/
}RTNODE; /*树结点的数据类型*/
RTNODE *root; /*树的根结点指针*/
```

3. 树的遍历

在应用树结构时，常要求按某种次序获得树中全部结点的信息，这可通过树的遍历操作来实现。有以下几种常用的树的遍历方法：

（1） 树的前序遍历：首先访问根结点，然后从左到右按前序遍历根结点的各棵子树。

（2） 树的后序遍历：首先从左到右按后序遍历根结点的各棵子树，然后访问根结点

（3） 树的层次遍历：首先访问处于 0 层上的根结点，然后从左到右依次访问处于 1 层上的结点处于 2 层上的结点等等，即自上而下从左到右逐层访问树各层上的结点。

（4） 访问树中所有叶子结点。有时，树除叶子结点外，其余结点只表示数据之间的某种分支或层次关系，对应的数据信息全存储的叶子结点中，对于这种情况就可能要用这样的遍历方法。

按上述遍历的定义，得到图 8.3 所示树的各种遍历如下：

前序遍历的结点序列：1 2 5 6 3 4 7 8 9
后序遍历的结点序列：5 6 2 3 8 9 7 4 1
层次遍历的结点序列：1 2 3 4 5 6 7 8 9
访问叶子结点的序列：5 6 3 8 9

设树按标准存储形式存储，树的次数和树的结点类型与前面所述相同。下面给出树的前序遍历和层次遍历的函数定义。

因树的前序遍历的定义是递归的，非常适宜用递归函数实现，见下面的函数定义：

【函数】前序遍历（用递归实现）。

```
Void re-preorder(
     TNODE *t, /*树根指针*/
     Int m) /*树的次数*/
{     int I;
     if(t!=NULL){
     printf("%c",t->data); /*访问树结点*/
     for (I=0;I<m;I+ +) /*前序遍历各子树*/
     re-preorder(t->childp[i],m);
     }
}
```

如不用递归，可引入栈记录当前未访问完的子树的根结点指针。见下面的函数定义。

【函数】前序遍历（采用栈）

```
#define SN 100
void st-preorder(TNODE*t,int m)
{     TNODE * s[SN]; /*栈*/
     int top=0, /*栈顶指针，初始时，为空栈*/
     I; /*工作变量*/
     If (t= =NULL)return; /*空指针情况，立即返回*/
     S[top+ +]=t; /*树的根结点指针进栈*/
```

```
     While (top >0){ /*栈中有未处理完的子树，继续循环*/
          T=s[- -top]; /*取出栈顶子树的根结点*/
          Printf ("%c",t->data); /*访问子树的根结点*/
     /*子树的子树按逆序进栈，例它们以后能顺序出栈*/
     For (I=m-1;I>=0;I- -)
          If (t->child [i]!=NULL)
               S[top+ +]=t->child[i];
     }
}
```

下面考虑设计层次遍历的函数。因层次遍历时，若某结点被访问，则该结点的子结点应登录，等待被访问。顺序访问各层次上的结点，直至不再有未访问过的结点。为此函数应引入队列，

存储已被访问结点的子结点。假设函数不考虑队列溢出情况，并令队首和队尾指针分别为队首结点和下一个进队结点在队列数组中的存储元素的下标。层次遍历函数定义如下：

[函数]层次遍历

```
#define QN 100
int lev_order(TNODE * t ,int m )
{     TNODE *q[QN];
     TNODE *p;
     Int head=0, / * 下一个出队结点的存储下标 * /
          Tail=0, / * 下一个进队结点的存储下标 * /
          I; / * 工作变量 * /
     Q[tail++]=t; / * t 进队形 * /
     While (head !=tail){ / * 队列非空 * /
          P=q[head]; / * 取出队首结点 * /
          Head=(head+1)%QN;
          Printf("%c",p->data); / * 防问队首结点 * /
          For(I=0;I<m,I++) / * 被访问结点的子结点顺序进队 * /
          If(p->child[I]!=NULL){
               If(tail+1)%QN==head)return 0;
               Q[tail]=p->child[i];tail=(tail+1)%QN;
          }
     }
     return 1;
}
```

8.6.2 二叉树

1. 二叉树的基本概念

与一般的树结构比较，二叉树在结构上更规范和更有确定性，因此，应用也比树列为广泛。图 8.4 是一棵二叉对的例子，二叉树的每个吉点中以有两棵子二叉树，分别简称为左子树和右子树。因二叉树可以为空，所以二叉树中的结点可能没有子结点，可能只有一个左子结点，可能只有一个右子结点，也可能同时有左右两个子结点。

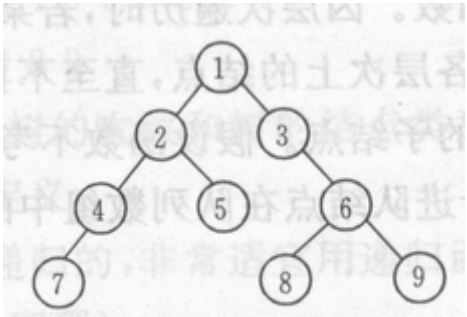


图 8.4 一棵二叉树

二叉树定义为：二叉树是一个有限的结点集合，该集合或者为空，或者由一个根结点及其两棵互不相交左、右二叉子树所组成的。

二叉树与树不同，首先二叉树可以为空，空的二叉树没有结点；另外，在二叉树中，结点的子树是有序的，分左、右两棵子二叉树。

一般情况下，二叉树常采用类似树的标准存储形式来存储。如结点只含字符的二叉树，其结点类型可以定义如下：

```
typedef struct node2{ / * 标准存储结构 * /
     char data;
     strucrt node2 * lchild;
     struct node2 * rchild;
}TNODE2;
```

2. 二叉树的遍历

除树的所有遍历方法也同样适用于二叉树外，二叉树还有中序遍历方法。现将其中二叉树的前序遍历、中序遍历和后序遍历方法一起列出如下：

（1） 前序遍历：

访问根结点，
按前序遍历根结点的左子树，
按前序遍历根结点的右子树。

（2） 中序遍历：

按中序遍历根结点的左子树，
访问根结点，
按中序遍历根结点的右子树。

（3） 后序遍历：

按后序遍历根结点的左子树，
按后序遍历根结点的右子树，
访问根结点。

对于图 8.4 所示的二叉树，前序遍历、中序遍历和后序遍历结果分别如下：

前序遍历：1 2 4 7 5 3 6 8 9
中序遍历：7 4 2 5 1 3 8 6 9
后序遍历：7 4 5 2 8 9 6 3 1

以上三种遍历方法都是递归定义的，实现上述遍历的递归函数分别定义如下：

[函数]前序遍历。

```
Void re_preorder(TNODE2 * t)
{     if (t! =NULL){ / * 非空二叉树 * /
      printf("%c",t->data);
      re_preorder(t->lchild);
      re_preorder(t->rchild);
      }
}
```

[函数]中序遍历。

```
Void re_midorder (TNODE2 * t)
{ if (t!=NULL){ / * 非空二叉树 * /
      re_midorder(t-lchild);
      prinf("%c",t->data);
      re_midorder(t->rchild);
      }
}
```

[函数]后序遍历.

```
Void re_posorder(TNODE2 * t)
{     if(t !=NULL){ / * 非空二叉树 * /
      re_posorder(t->lchild);
      re_posorder(t->rchild);
      printf("%c",t->data);
```

```
     }
}
```

如不用递归方法编写，可引入栈。以中序遍历为例，使用链接栈编写的中序遍历函数。定义如下：

[函数]中序遍历（非递归，采用链接栈）。

```
Typedef struct snode{                    / * 栈元类型 * /
     TNODE2 * tnodept;                   / * 栈结点为指向二叉结点的指针 * /
     Struct snode * link;                / * 栈结点链接指针 * /
}SNODE;
void st_midorder(TNODE2 * t)
{     SNODE * top=NULL,                  / * 栈顶指针 * /
          * p;                           / * 工作变量 * /
     while(t!=NULL||top !=NULL){         / * 二叉树未访问完，或栈非空 * /
          while(t!=NULL){                / * 一棵二叉树还未访问完，循环 * /
/ * 根结点进栈 * /
     p=(SNODE
*)malloc(sizeof(SNODE));
     p->tnodept=t;
     p->link=top;
     top=p;
/ * 沿着左子树前进，将经过的结点依次进栈 * /
     t=t->lchild;
}
if (top!=NULL){                          / * 如栈非空 * /
     t=top->tnodept;                     / * 栈顶点出栈 * /
     p=top;
     top=top->link;
     free(p);                            / * 释放栈顶点的空间 * /
     printf("%c",t->data);               / * 访问根结点 * /
     t=t->rchild;                        / * 向右子树前进 * /
     }
  }
}
```

8．6．3 二叉查找树

在有序的顺序存储的线性表中寻找给定键值的结点，二分查找是一个很好的方法。如果结点插入和删除又相当频繁的情况，采用线性表并顺序存储是很不理想的。希望有一种数据结构，同时具有链接存储便于增删和又能像二分查找那样实行快速查找，其中要查找树就是一种理想的数据结构。

查找树 T 是一棵二叉树，它或者是空，或者满足以下三个条件；

（1） 如果 T 的根结点的左子树非空，其左子树所有结点的键值都小于 T 的根结点的键值；

（2） 如果 T 的根结点的右子树非空，其右子树所有结点的键值都大于 T 的根结点的键值；

（3） T 的根结点的左子树和右子树也都是查找树。

按查找树的定义推知，如中序遍历查找树，就能得到一个排好序的结点序列。查找树上有以下三种基本操作：查找、插入和删除。设查找树的结点只存储结点的键值，其类型与前面的二叉树的结点类型相同。

1. 静态查找

查找树上的静态查找是在查找上找键值为 key 的结点是否存在，这可按以下步骤在查找树 T 上找值为 key 的结点：

（1） 如果查找树 T 为空二叉树，则查找失败，结束查找；

（2） 如果查找树的根结点的键值等于 key ，则查找成功，结束查找；

（3） 如果 key 小于根结点的键值，则沿着根结点的左子树查找，即根结点的左子树作为新的查找树 T 继续查找；

（4） 如果 key 大于根结点的键值，则沿着根结点的右子树查找，即将根结点的右子树作为新的查找树 T 继续查找。

按上述步骤编写查找函数如下：

[函数]静态查找函数

```
TNODE2 *search(
     TNODE2 *t,                      / * 查找树根结点指针 * /
    Char key)
{   while (t!=NULL)
     if(t->date==key)return t;       / * 找到 * /
else if (key<t->data)t=t->lchild;    / * 沿左子树查找 * /
          else t=t->rchild;          / *沿右子树查找 * /
return NULL;                         / * 未找到 * /
```

```
}
```

2. 动态查找

在查找树上，为插入和删除操作需要而使用的查找称为动态查的，动态查的应得到两个指针，一个指向键值为 key 的结点，另一个指向该指点的父结点。为此查找函数就设 4 个参：查找树的根结点指针 t，查找值 key，存储键值为 key 结点的父结点的指针变是的指针 pkpt，存储键值为 key 结点的指针变量的指针 kpt。但函数要考虑以下几种不同情况：

（1） 因查找树为空，查找失败，函数使

```
* kpt=NULL, * pkpt=NULL;
```

(2)因查找树中没有键值为 key 的结点，函数一直寻找至查找路径的最后一个结点，* pkpt 指向该结点，* kpt=NULL，如果插入键值为 key 的结点，就插在该结点下；

（3）查找成功，*kpt 指向键值为 key 的结点，* pkp 指向它的父结点。
下面是实现上述要求的动态查找函数定义：

[函数]动态查找函数。

```
Void search(TNODE2 t,              / * 查找树的根结点指针 * /
     Char key,                     / * 查找值 * /
     TNODE2 * * pkpt,              /*key 结点的父结点的指针的指针 * /
     TNODE2 * * kpt)               / * key 结点的指针的指针上* /
{ * pkpt=NULL;
     *kpt=t;
     while(* kpt!=NULL){
        if(*
kpt)->data==key)return             / * 找到 */
          * pkpt=* kpt;            / * 当前结点作为父结点，继续查找 * /
/*按查找树的性质，确定查找路径：
* /
          if (key<(*kpt)->data)*
kpt=(*kpt)->lchild;
     else *kpt=(* kpt)->rchild;
     }
}
```

如有指针变量 rpt 指向查找树的根结点，若希望用指针变量 q 指向查到的结点，指针变量 p 指向找到结点的父结点，要找键值为 x 的结点，可以用以下函数调用：

search(rpt,'x',&p,&q)

3. 查找树上插入结点

利用动态查找函数,在查找树上插入一个键值为 key 的结点的函数可按以下思想设计：首先利用动态查找函数确定新结点的插入位置，然后分以下几种情况作为相应的处理，如相同键值的结点已在查找树中，则不再插入；如查找树为空树，则以新结点为查找树;最后根据要插入结点的键值与插入后的父结点的键值比较就能确定新结点是父结点的左子结点，或为右子结点，并作相应插入。见下面的函数定义：

[函数]查找树上插入新结点。

```
/ * 插入成功，函数返回 0，否则函
数返回 1 * /
int insert(TNODE2 * * pt,char key)
{   TNODE2 p, * q ,* r;
     search(* pt,key,&p,&q);     / * 寻找插入位置 * /
     if (q!=NULL)return 1;       / * 已在树中，不再插入 * /
     r=(TNODE2
                                 / * 新结点 * /
* )malloic(sizeof(TNODE2));
     r->data=key;
     r->lchild=r->rchild=NULL;
                                 /* 如为空树,新结点为查找树 *
     if (p==NULL) * pt=r;        /
     else if (p->data>key)
          p->lchild=r;           / * 作为左子结点 * /
     else p->rchild=r;           / * 作为右子点 * /
return 0;                        / * 插入成功 * /
}
```

4. 查找树上删除结点

删除查找树上键值为 key 的结点的操作可按以下步骤进行：

（1） 调用查找函数确定被删结点位置；

（2） 如被删结点不在树上，则函数返回；否则按以下多种情况作结点删除处理：

• 被删结点是根结点，又分以下两种情况作不同处理：被删结点无左子树，则以被删结点的右子树作为删除后的树；被删结点有左子树，则用被删结点的左子结点作为根结点,并把被删结点的右子树作为被删结点的左子树按中序遍历的最后一个结点的右子树。

·被删桔点不是根结点，且被删结点无左子结点：被删结点是它的父结点左子结点，则把被删结点的右子树作为被删结点的父结点的左子树；；被删结点是它结点的右子结点，则把被删结点的右子树作为被删结点的父结点的右子树；

·被删结点不是根结点，且被删结点无左子结点，则被删结点的右子树作为被删结点的左子树按中序遍历的最后一个结点的右子树，同时进行以下操作：如被删结点是它的父结点的左子结点，则把被删结点的左子树作为被删结点的父结点的左子树；如被删除结点是它的父结点的右子结点，则把被删结点的左子树作为被删结点的父结点的右子树。

以下是按上述步骤编写的删除查找树键值为 key 的结点的函数：

[函数]删除查找树键值为 key 的结点。

```
Int delete(TNODE2 * *pt,char key)
{     TNODE2 * p,*q,*r;
     search(*pt,key,&p,&q);
     if(q==NULL)return 1;
     if(p==NULL) / * 被删结点是根结点 * /
          if(q->lchild==NULL)
/ * 被删结点无左子树，被删结点的右子树作为删除后的树 * /
               pt=q->rchild;
          else{ /*被删结点有左子树用被删结点的左子点作为根结点 * /
               * pt=q->lchild;
/ * 寻找被删结点的左子树按中序最后一个结点 * /
               r=q->lchild;
               while(r->rchild!=NULL)
                    r=r->rchild;
/ * 被删结点的右子树作为找到结点的右子树 * /
               r->rchild=q->rchild;
          }
     else if (q->lchild==NULL)
/*被删结点不是根结点,且被删结点无左子结点*/
        if(q==p->lchild)
/ * 被删结点是它的父结点的左子结点，把被删结点的右子树作为被删结点的父结点的左子树木* /
          p->lchild=q->rchild;
     else
/*被删结点是它的父结点的右子结点,把被删结点的右子结子树作为被删结点的父结点的右子树*/
           p->rchild=q->rchild;
     else{
/ *被删结点不是根结点，且被删结点有左子结点 * /
/ * 寻找被删结点的左子树按中序遍历的最后一个结点 * /
     r=q->lchild;
```

```
     while(r->rchild!=NULL)r=r->rchild;
/ *被删结点的右子树作为找到的结点的右子树 * /
     r->rchild=q->rchild;
     if(q==p->lchild)
/* 被删结点是它的父结点的左子结点，把被删结点的左子树作为被删结点的父结点的左子树 * /
          p->lchild=q->lchild;
     else
/* 被删结点是它的父结点的左子结点，把被删结点的左子树作为被删结点的父结点的右子树 * /
          p->rchild=q->lchild;
          }
     free(q);
     return 0;
}
```

8.6.4 穿线树

当用标准形式存储二叉树时,树中有一半以上的指针是空指针,将空指针用来穿线,是一种实用的做法.

设有一棵采用标准形式存储的二叉树T,对于T中的每个结点K,如它没有左(或右)子结点,而K1是K的按中序遍历的前面(或后面)结点,则置结点K的左(或右)指针为K1结点的指针.为了与K结点的真正子结点指针区别,另需在结点上增加两个标志字段如lflg和rflg。当标志值为0。经如此改造后的二叉树T为中序穿线树，简称穿线树。设穿线树的结点类型定义为

```
typedef stuct node{ / *穿线树结点类型 * /
     char data;
     struct node * lchild, * rchild ;
     int lflg, rflg;
}LTNODE;
```

穿线树除有二叉树根结点指针外，还需要一个按中序遍历第一个结点的指针。以下是关于穿线树上操作的一些常用函数。

1. 找出指定点的中序遍历的前驱结点

设指定结点为t，为找出结点t 的中序遍历前驱结点,有几种情况需要考虑。如当前结点已前驱穿线，则按其左指针能立即得到它的前驱结点；如当前结点是中序遍历的首结点，则它没有前驱结点；否则它的前驱结点是它的左子树中的最右叶子的结点。

[函数]找出由指针t所指结点的中序遍历的前驱结点。

```
LTNODE * pred(LTNODE * t)
{     if (t->lflg==1||t->lchild ==NULL)
/ * 或者当前结点前驱穿线，或者当前结点是中序遍历首结点 * /
          return t->lchild;
     t=t->lchild; / *向前结点的左子结点前进一步 * /
     while(t->rflg==0)
          t=t->rchild; / * 沿着右子结点前进，直至叶结点 * /
     return t;
}
```

2.找出指定结点的中序遍历的后驱结点

设指定结点为 t，找结点 t 的中序遍历后继结点，也同样有几种情况需要考虑。如当前结点已后继穿线，则按其右指针能立即得到它的后继结点；如当前结点是中序遍历的未结点,则它没有后继结点；否则它的后继结点是它的右子树中的最左叶子结点。

【函数】找出由指针 t 所指结点的中序遍历的后驱结点。

```
LTNODE * succ(LTNODE * t)
 {     if(t->rflg==1||t->rchild==NULL)
/ * 或者当前结点后继穿线,或者当前结点是中序遍历末结点 * /
               return t->rchild ;
          t=t->rchild; / * 向当前结点的右子结点前进一步 * /
          while(t->lflg==0)
               t=t->lchild; / * 沿着左子结点前进,直至叶子结点 * /
          return t;
}
```

3.插入新结点,使新结点是某参考结点的中序遍历的前驱结点

首先函数需设三个参数,待插入结点指针 q,插入参考结点指针 p,另因新结点插入后,它可能会成为中序遍历的首结点,函数需要穿插线树中序遍历首结点指针的指针 hpt.插入时需考虑以下几种情况. P 所指结点已前驱穿线,或 p 所指结点是中序遍历的首结点,则 q 结点应成为 P 结点的左子结点,且 q 结点的左穿线是原 p 结点的左穿线,q 结点的右穿线指向 p 所指结点.另对于原 p 所指结点是中序遍历首结点情况,插入后 q 所指结点成为首结点,应修改 hpt 所指的指针,当 p 所指结点有左子树时,则先要找出它的前驱结点,并插在它的前驱结点的右面.细节见函数 left_insert()

[函数]在指定结点之前插入新结点.

```
Void left_insert(LTNODE * p,LTNODE * q,LNODE * * hpt)
{     LTNODE * temp;
     if(p->lchild==1||p->lchild==NULL){
```

```
          if(p->lchild==NULL)*hpt=q;
          q->lflg=p->lflg;q-lchild=p->lchild;
          q->rflg=1;q->lchild=p;
          p->lflg=0;p->lchild=q;
 }else{
          temp=pred(p);
          q->rflg=temp->rflg;q->rchild=temp->rchild;
          q->lflg=1;q->lchild=temp;
          temp->rflg=0;temp->rchild=q;
     }
}
```

4.插入新结点,使新结点是某参考结点的中序遍历的后继指点

这里的函数只需设二个参数,待插入结点指针 q 和插入参考结点指针 p.插入进需考虑的情况与函数 left_insert()类似,细节见函 right_insert().

```
[函数]在指定结点之后插入新结点.
Void right_insert(LTNODE * p,LTNODE *q)
{     LTNODE * temp;
     if(p->rflg==1||p->rchild==NULL){
     q->rflg=p->rflg;q->rchild=p->rchild;
     q->lflg=1;q->child=p;
     p->rflg=0;p->rchild=q;
 }else{
     temp=succ(p);
     q->lflg=temp->lflg;q->lchild=temp->lchild;
     q->rflg=1;q->rchild=temp;
     temp->lflg=0;temp->lchild=q;
     }
}
```

5.利用前述函数,对给定的字符串,建立一棵查找穿线树

```
[函数]LTNODE * th_sort_t(char * str, LTNOE * * hpt)
{     LTNODE * root, * head,* p,* t;
     if(* str=='\0')retur NULL;
/ * 建立根结点 * /
     head=root=(LTNODE * )MALLOC (sizeof(LTNODE)) ;
     root->data=* srt ++;
     root->lchild=root->rchild=NULL;
     root->lflg=root->rflg=0;
     while(* str){ / * 字符串未结束, 循环 * /
          t=(LTNODE * )MALLOC(sizeof(LTNODE));
          t->data= * str++; / * 建立新结点 * /
```

```
          p=root; / * 找插入位置 * /
          while(1){ / * 找插入位置循环 * /
               if (t->data<=p->data)
                    if (!p->lflg && p->lchild)
                         p=p->lchild; / * 向左子树寻找 * /
                    else{left_insert(p,t,&head); / * 插在* p 之左 * /
                              break;
                              }
               else if (!p->rflg &&p->rchild)
                              p=p->rchild; / * 向右子树寻找 * /
                         else{right_insert(p,t); / * 插在 * p 之右 * /
                              break;
               }
          }
     }
     * hpt=head; / * 回送中序遍历首结点指针 * /
     return root; / * 返回二叉树的根指针 * /
}
```

8．6．5 树形数据结构程序设计实例

[例 8．13]已知字符列的前 n 个元素，建立一棵二叉树。

设字符列首元素指针为 str,根结点为 str[i],则它的左、右子结点分别为 str[2 * I+1]和 str[2 * I+2]，采用递归方法构造，参见以下函数。

[函数]以 str[I] 为根，左、右子结点分别为 str[2 * I+1]和 str[2 * I+2].

```
TNODE2 * build(char * str,int I,int n)
{ / * 已知字符列，构造一棵二叉树 * /
     TNODE2 * P;
     If (I>=n)return NULL;
     P=(TNODE2*)malloc(sizeof(TNODE2));
     P->data=str[i];
     p->lchild=build(str,2*I+1,n);
     p->rchild=build(str,2*I+1,n);
     return p;
}
```

若设字符列首元素指针为 str，欲构造二叉树的元素为 str[i]至 str[j],令根据结点为 str[(I+j)/2],它的左子树结点为 str [I]至 str[(I+j)/2-1],右子树结点 str [(I+j)/2+1]/str[j]，也采用递归方法构造，就有以下函数。

【函数】中间为根，前半元素为左子树，右半元素为右子树。

```
TNODE2 * build(char * str ,int I,int j)
{ /*已知字符列，构造一棵二叉树*/
     TNDE2*P;
     If (I>j)return NULL;
     P=(TNODE2*)malloc(sizeof(TNODE2));
     p->data=str[(I+j)/2];
     p->rchild=build(str,I,(I+j)/2-1);
     p->rchild=build(str,(I+j)/2+1,j);
     return p;
}
```

【例 8.14】编制已知二叉树的前序遍历和中序遍历序列，构造对应二叉树的函数。

由二叉树的前序遍历和中序遍历能够惟一确定一棵二叉树。用示例说明构造方法，设欲构造的二叉树的前序和中序遍历结果分别为

A,B,D,E,C,F,G,和 D,B,E,A,C,G,F

首先，由前序遍历序列，其中结点 A 为二叉树的根，再由中序遍历序列知，A 前面的 D，B 和 E 一定是根结点 A 的左子树上的结点，A 后面的 C，G 和 F 是 A 的右子树上的结点。对 A 的左子树和右子树再分别由对应子树前序遍历序列的首字符确定子树的根结点，由对应的中序遍历序列中的根结点位置分别确定构成它们的左子树和右子树。依此类推，确定二叉树的全部结点，最终构结出二叉树。下面是采用递归方法编写的函数。

[函数]由二叉树的前序遍历和中序遍历序列，构造对应的二叉树。

```
TBIDE2 * restore(char * pstr,char * istr,int n)
{     TNODE2 * ptr;
     char * rstr;
     int k;
     if(n<=0)return NULL;
     ptr=(TNODE2 * ) malloc (sizeof(TNODE2));
     ptr->data=* pstr;
     for(rstr=istr;rstr<istr+n;rstr++)
          if(* rstr== * pstr )break; / * 在中序遍历序列中找根结点 * /
     k=rstr-istr; / * 左子树的结点数 * /
     ptr->lchild=restore(pstr+1,istr,k);
     ptr->rchild=restore(pstr+k+1,rstr+1,n-k-1);
     return ptr;
}
```

[例 8. 15]已知二叉树，将其改建为中序穿线树。

将二叉树改建成穿线树，就是将结点中的空指针让它指向前驱或后继结点。具体实现方法是对二叉树进行中序遍历，在遍历过程中，如当前被访问结点的左子结点指针为空，则让它指向当前结点的前驱结点；如它的前驱结点的右子结点指针为空，则让它指向当前结点。下面采用链接栈改建二叉树为中序穿线树的函数，其中栈结点的类型需改为 LTNODE。

[函数]

```
LNODE * top=NULL;
LNODE * threaded(LTNODE * t )
{     LTNODE * p, / * 指向当前结点 * /
                    * pr / * 指向当前结点的前驱结点 * /
                    * h; / * 暂在中序遍历的首结点指针 * /
      if(t==NULL)return NULL;
      top=NULL; / * 置空栈 * /
      p=t;pr=NULL;
      do {
           for(;p!=NULL;){
                l_push(p,&top); / * 结点前栈，待处理后进入其右子树 * /
                p=p->lchild; / * 沿着左子树前进 * /
           }
           l_pop(&p,&top);
           if (pr==NULL){
                h=p; / * p 指向最左结点，是中序遍历首结点* /
                p->lflg=0; / * 继续保持空指针 * /
           }else{
                if(p->lchild==NULL){
                     p->lchild=pr;p->lflg=1; / *向前穿线 * /
                }
           else p->lflg=0;
           if(pr->rchild==NULL){
                pr->rchild=p;
                pr->rflg=1; / * 向右穿线 * /
           }
           else pr->rflg=0;
      }
      pr=p;
      p=p->rchild; / *进入右子树 * /
      }while((p!=NULL)||top!=NULL;
      pr->rflg=0; / * 最右结点继续保持空指针 * /
      return h;
}
```

[例 8．16]编写判一棵二叉树是否是一棵查找树的函数。

设二叉树的数据类型为：

```
typedef struct node{
               int val;
               struct node * lchild,* rchild;
}NODE;
```

二叉查找树的所有子树有性质：子树根结点的值都比它的左子树上的结点的值大，而又都比它的右子树上的结点的值小。采用递归方法，对每个结点分别求出它的左、右子树的最大结点值和最小结点值。对于左子树，它的最大结点值应比根结点的值小；它就是一棵查找二叉树；否则，它不是二叉查找树。细节见以下程序中的函数定义。

[例 8. 13 程序]

```
#include<stdio.h>
#include<stdlib.h>
int a[]={1,2,3,4,5,6,7}; / * 作为测试数据 * /
typedef struct node {
     int val;
     strct node * lchild,* rchild;
}NODE;
void in Trave(NODE * t) / * 只为了检查程序的正确性 * /
{     if (t){inTrave(t->lchild);
                    printf("%4d",t->val);
                    inTrave(t->rchild);
          }
}
NODE * built(int a[],int low,int high)
{     NODE * p;int mid;
     if (low>high)return NULL;
     mid=(high+low)/2;
     p=(NODE * )malloc(size(NODE));
     p->val=a[mid];
     p->lchild=built(a,low,mid-1);
     p->rchild=built(a,mid+1,high);
     return p;
}
/ * 判二叉树是查找树的函数 * /
int inSeachtree(NOCDE * t,int * minp,int * maxp)
{     int lmin,lmax,rmin ,rmax;
     if (t==NULL)return 1;
     lmin=lmax=t->val;
     if(t->lchild){
          if(isSearchtree(t->lchild,&lmin,&lmax)==0)return 0;
```

```
            if(t->val<=lmax)return 0;
            }
      rmin=rmax=t->val;
      if(t->rchild){
            if(is Searchtree(t->rchild,&rmin,&rmax)==0)return 0;
            if(t->val>=rmin)return 0;
      }
      * minp=lmin; * maxp=rmax;
      return 1;
}
void main()
{     int I,min,max;
      NODE * r;
      for(I=0 ;I<7;I++)
            printf("%4d",a[i]);
            printf("\n");
            r=built(a,0,6);
      inTrave(r);
      printf("\n");
      I=isSearchtree(r,&min,&max);
      Printf("\t%s\n",(I==1)? "OK":"NO");
}
```

8.7 图

图是比树形结构更复杂的一种数据结构。在线性结构中，除首结点没有前驱，末结点没有后继之外，一个结点只有一个前驱结点和一个后继结点。在树形结构中，除根结点没有前驱结点外，一个结点虽只有一个前驱结点，但可以有多个后继结点。在图结构中，一个结点的前驱结点和后继结点的个数则是任意的。

8．7．1 图的基础知识

1． 图的基本概念

在图中，数据结构中的结点被称为顶点，结点之间的关系被称为边。一个图 G 由非空有限的顶点集合 V 和有限的边的集合 E 组成，记为 G=（V，E）。图 8.5 是两个图的例子，图分有向图和无向图两种，其中图 8.5（a）是无向图，图 8.5（b）是有向图。无向图的边是顶点的无序偶，用（i,j）来表示顶点 i 和 j 之间的边。在无向图 G 中，如果 i≠j,i、j∈V,(i,j)∈E,即 i 和 j 是 G 的两个不同的顶点，（i,j）是 G 中一条边，顶点 i 和顶点 j 相邻顶点，边（i,j）是与顶点

i 和 j 相关联的边。

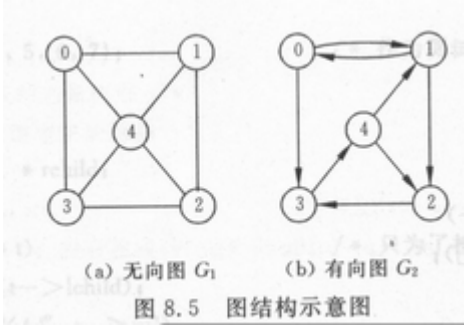


图 8.5 图结构示意图

有向图的边是项点的有序偶，有向图的边也称为弧，用<i,j>来表示顶点 i 和 j 之间的弧。在有向图的 G 中，如果 i ≠j,i、j∈V,<i,j>∈E，即 i 和 j 是 G 的两上不同的顶点，<i,j>是 G 中的一条边，称顶点 i 是这条边的尾顶点，顶点 j 是这条边的头项点，称顶点 i 邻接到顶点 j，顶点 j 邻接于顶点 i，弧<i,j>是与顶点 i 和 j 相关联的弧。

如果限定任何一条边或弧的两个顶点都不相同，一定有 n 个顶点的无向图至多有 n(n-1)/2 条边，这样无向图称为无向完全图。一个有向图至多有 n(n-1)条弧，这样有向图称为有向完全图。

如果同为无向图或同为有向图的两个图 G1=（V1，E1）和 G2=(V2,E2)满足

$$V2 \subseteq V1 \quad E2 \subseteq E1$$

则称图 G2 是图 G1 的子图。

在无向图中，一个顶点的度等于与其相邻接的顶点个数。在有向图中，一个顶点的入度等于邻接到该顶点的顶点个数，其出度等于邻接于该顶点的个数。

在图 G=（V，E）中，如果存在顶点序列（v0,v1,…,vk）其中 V0=p,vk=q,且（v0=p,vk=q,）都在 E 中，则称顶点 P 到顶点 Q 有一条路径，并用表示这条路径，路径的长度是路径的边数，这条路径的长度为 0。若 G 是有向图，则路径也是有向的。

如果无向图 G 中任意两个不同顶点 i 和 j 之间都有从 i 到 j 的路径，则称无向图 G 是连通的。称无向图 G 中的极大连通子图为图 G 的连通分量。

对于有向图，如果有向图 G 中任意两个不同顶点 i 和 j 之间都有一条从 i 到 j 的路径，同时还有一条从 j 到 i 的路径，则称有向图 G 是强连通的。称有向图 G 中的极大强连通子图为图 G 的强连通分量。
如果给图的每条边赋予一个实数作为边的权，则该图为带权图，或简称为网。

2. 图的存储结构

最常用的图的存储结构是邻接矩阵和邻接表。

（1） 邻接矩阵

图的邻接矩阵是反映顶点间邻接关系的矩阵，设 G=（V，E）是具有 n(n≥1)个顶点的图，G 的邻接矩阵 M 是一个 n 行 n 列的矩阵，并有若（i,j）或<i,j>∈E,则 M[i][j]=1；否则，M[i][j]=0。例如，图 8.5 的图 G1 和图 G2 的邻接矩阵分别如图 8.6 的 M1 和 M2 所示。

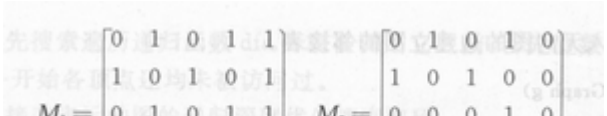


图 8.6 邻接矩阵

由邻接矩阵的定义知，无向图的邻接矩阵是对称的，有向图的邻接矩阵就不一定对称。从邻接矩阵很容易判定图中的两个顶点是否相邻接，并可方便地求得各个顶点的度。对于无向图，其邻接矩阵第 i 行元素的和即为顶点 i 的度。对于有向度，其邻接矩阵的 i 元素之和顶点 i 的出度，而邻接矩阵的第 i 列元素之和为顶点 j 的入度。

如果图 G=（V，E）是一个网，若（i,j）或<i,j>属于 E，则邻接矩阵中的元素 M[i][j]为（i,j）或<i,j>上的权，这里约定所有的权均不小于 0。若（i,j）或<i,j>不属于 E，则 M[i][j]为无穷大或为大于图中任何权值的实数。

若 n 个顶点的图 G 用邻接矩阵表示，则图的数据类型可定义为：

```
#define → 30 / * 图中最多的顶点数 * /

typedef int Graph[N][N];
```

若 G 为网，则网的类型可定义成；

```
typedef double GraphNet[N][N];
```

(2)邻接表

在图的邻接表表示中，为图的每个顶点建立一个链表，且第 i 个链表中的结点代表与顶点 i 相关联的一条边或由顶点 i 出发的一条弧。有 n 个顶点的图，需用 n 个链表中表示这个 n 个链表的头指针通常有顺序线性表存储。

在无向图的邻接表中，对应某结点的链表的结点个数就是该顶点的度。

若图用邻接表来表示，其数据类型可定义如下；

```
#define → 30 / * 图中最多的顶点数 * /
typedef struct node { / *邻接表中链表的结点类型 * /
    int vno; / * 邻接顶点的顶点序号 * /
    struct node * next; / * 后继邻接顶点 * /
}edgeNode;
typedef edgeNode * Graph[N]; / * 邻接表类型 * /
```

下面输入无向图的边建立图的邻接表函数 creat_list，函数返回图的顶点数。

[函数]输入无向图的边建立图的邻接表。

```
int creat_list(Graph g)
{    int vn,en,k,i,j;
    edgeNode * p;
    printf("输入顶点数 \n ");scanf("%d",&vn);
    printf("输入边数\n"); scanf("%d",&en);
    for(k=0;k<vn;k++)g[k]=NULL; / * 置空邻接表 * /
    for(k=0;k<en;k++){ / * 输入 en 条边 * /
        printf("输入边：");
        scanf("%d%d,&i,&j);
        p=(edgeNode * )malloc(sizeof(edgeNode));
        p->vno=j;p->next=g[i];g[i]=p;
        p=(edgeNode * )malloc(sizeof(edgeNode));
        p->vno=i;p-next=g[j];g[j]=p;
    }
    return vn; / * 返回图的顶点数 * /
}
```

3. 图的遍历

图的遍历是指从图的某个顶点出发，沿着图的边或弧访问图的每个顶点，并且每个顶点只被访问一次。图的深度优先搜索遍历和图的广度优先搜索遍历是图最典型的两种遍历方法。

（1） 深度优先搜索遍历

图的深度优先搜索遍历的遍历过程是：首先选定一个出发顶点 v ,并访问之。接着选择一个与 v 相邻接并且未被访问过的顶点 w 访问之，于从 w 开始进地深度优先搜索遍历。每当到达一个其所有相邻接的顶点都已被访问过的顶点时，就从最近所访问的顶点开始依次回退到某个顶点，该顶点尚未曾访问过的邻接顶点，再从该邻接顶点开始继续进行深度优先优先搜索遍历。上述过程在两种可能情况下终止，或所有顶点已都被访问，或从任一个已被访问过的顶点出发，再也无法到达未曾访问过的顶点。

对于无向图，如果图是连通的，那么按深度优先搜索遍历时，可遍历全部顶点，得到全部顶点的一个遍历序列。如果遍历序列没有包含所有顶点，那么该图是不连通的。如对图 8.5 的 G1，从顶点 3 开始作深度优先搜索遍历，得到遍列序列为：3 0 1 2 4.对图 8。5 的 G2，从顶点 3 开始作深度优先搜索遍历，得到遍历序列则为：3 4 1 0 2。

图的深度优先搜索遍历递归函数 dfs 要求在调用它之前，首先置数组 visited 的全部元素为 0，表示从一开始各顶点还均未被访问过。

[函数]邻接表表示的图的递归深度优先搜索遍历。

```
int visited[N];
Void dfs(Graph g,int i)
{    edgeNode * t;
    printf("%4d",i); / *访问顶点 i * /
    visited[i]=1; / * 置 i 顶点已访问 * /
    t=g[i]; / * 检查所有与顶点 i 相邻接的顶点 * /
    while (t!=NULL){
        if(visited[t->vno]==0) / * 如该顶点未被访问过 * /
            dfs(g,t->vno); / * 从邻接顶点出发深度优先搜索 * /
        t=t->next; / * 考察下一人个邻接顶点 * /
    }
}
```

(2)广度优先搜索遍历

广度优先的遍历过程是：首先访问出发顶点 v，然后访问与顶点 v 邻接的全部未被访问过的顶点 w0,w1,…,wk-1 ；接着再依次访问与顶点 w0,w1,…,wk-1 邻接的全部未被访问过的顶点。依此类推，直至图的所有顶点都被访问到，或出发顶点 v 所在的连通分量的全部顶点都被访问到为止。如图 8.5 的 G1，从顶点 3 开始作广度优先搜索遍历，得到遍历序列为：3 0 2 4 1。对图 8.5 的 G2 从顶点开始作广度优先遍历，得到的序列为 3 4 1 2 0。

从广度优先搜索遍历过程知，若顶点 v 在顶点 w 之前被访问，则对 v 相邻的顶点的访问就先于只与 w 相邻的那些顶点的访问。因此需要一个队列来存放被访问过的顶点，以便按顶点的访问顺序依次访问这些顶点相邻接的其他还未被访问过的顶点。

设用邻接表示的图的广度优先搜索遍历.

[算法]邻接表表示的图的广度优先搜索遍历。

```
{    置全部顶点都未被访问过；
        队列置空；
        访问出发顶点，并置该顶点已被访问的标志；
```

```
        出发顶点进队；
        while（队不空）{ / * 在队列非空情况下循环 * /
            取队列首顶点于 v；
            按邻接表，顺序考察与顶点 v 邻接的各顶点 w
        if(顶点 w 未被访问过){/ * 一个未被访问过的顶点 * /
            访问顶点 w；
            置顶点 w 已被访问标志；
            顶点 w 进队；
        }
    }
}
```

以下是按上述算法编写的函数。

[函数]图的广度优先搜索遍历。

```
# define M(2 * N)
int queue[M];
void bfs(Graph g,int s ,int n)
{    int i,v,w,head,tail;
    edgeNode * t;
    for(i=0;i<n;i++)
            visited[i]=0; / * 置全部顶点都未被访问过 * /
    head=tail=0; / * 队列置空 * /
    printf("%4d",s); / * 访问出发顶点 * /
    visited[s]=1; / * 置该顶点已被访问的标志 * /
    queue[tail++]=s / * 出发顶点进队 * /
    while(head!=tail){ / * 队不空循环 * /
        v=queue[head]; / *取队列首顶点于 v * /
        head=(head+1)%M;
            for(t=g[v];t!=NULL;t=t->next){ / * 顺序考察与顶点 v 邻接的
各顶点 * /
                w=t->vno;
                if(visited[w]==0){ / *顶点 w 未被访问过 * /
                    printf("%4d",w); / * 访问顶点 w * /
                    visited[w]=1; / * 置顶点 w 已被访问标志 * /
                if(tail+1)% M==head){
                    printf("队理满，结束程序");
                }
                queue[tail]=w; / * 顶点 w 进队 * /
                tail=(tail+1)%M;
            }
        }
```

```
    }
}
```

以下代码能利用前面的类型定义和函数，输入图的边，建立图的邻接表，并进行深度优先和广度优先搜索遍历图。

```
Void main()
{    Graph g;int n;i;
    n=creat_list(g);
    for(i=0;i<n;i++)visited[i]=0;
    dfs(g,0);printf("\n");
    bfs(g,0,n);printf("\n");
}
}
```

8.7.2 最小代价生成树

设 G=（V，E）是一个连通的无向图，若 G1 是包含 G 中所有顶点的一个无回路的连通子图，则称 G1 为 G 后棵生成树。从 G 的任一顶点出发进行一次深度优先或广度优先搜索遍历图 G，由遍历过程中走过的边的集合和访问过的顶点集合构成的子图是一个连通子图，这就是图 G 的一棵和成树。

含有 n 个顶点的连通图的生成树有 n 个顶点和 n-1 条边。对一个带权的图（网），在一棵生成树中，各条边的权植之和称为这棵生成树的代价。一个图可以有许多不同的生成树，其中代价最小的生成树称为最小代价生成树。

以下讨论求连通的带权无向图的最小代价生成树的 Prim 算法。设已知 G=（V，E）是一个带权连通无向图，顶点 V={0,1,2,…,n-1}。设 U 是构造生成树过程中已被考虑在生成树上的顶点集合。初始时，U 只包含一个出发顶点。设 T 是构造生成树过程中已被考虑在生成树上的顶点集合，初始时 T 为空。如果边（i,j）具有最小代价，且 i ∈U，j∈v-U,那么最小代价生成树应包含边（i,j）.把 j 加到 U 中，把（i,j）加到 T 中。重复上述过程，直到 U 等于 V 为止。这时，T 即为要求的最小代价生成树的边的集合。

设用邻接矩阵表示图,为了便于程序处理。还需进一步引入一些辅助变量。数组 closest 用来存放顶点号，数组 lowcost 用来存放顶点号，数组 lowcost 用来存放代价。数组成 lowcost 存放 U 中顶点到 V-U 中顶点到 V-U 则顶点 i 在 V-U 中，并且顶点 closest[i]在 U 中与顶点 i 邻接的边中代价最小的边，其代价为 lowcost；若[i]lowcost[i]=∞，则表示 closest[i]与顶点 i 之间没有边。细节见以下函数。

[函数]用 Prim 算法生成最小代价生成树。

```
#define MAX_iNT 10000 / * 表示无穷大值 * /
#define n 100 / * 最多顶点个数 * /
```

```
int prim( / * 函数返回生成树的代价 * /
    int w[][N], / * 邻接矩阵 * /
    int n, / * 图的顶点数 * /
    int v;, / * 起始顶点 * /
    int tree[][3] / * 存放生成树的边 * /)
{    int i,j,k,p,min,c;
    int lowcost[N],closest[N];
    for(i=0;i<n;i++){
        closest[i]=v; / * U 中只有顶点 V * /
        lowcost[i]=w[v][i]; / * U 中顶点到 V-U 中各顶点的代价 * /
    }
    c=0;p=0;
    for(c=i=0;i<n-1;i++){ / * 寻找 n-1 条边，加到生成树 * /
        min=MAX_iNT;  / * 找最小权的边 * /
    for(j=0;j<n;j++) / *在从 U 到 V-U 中的最小权的边 * /
        if(lowcost[j]!=0&&lowcost[j]<min){ / * 找到权列小的边 * /
            min=lowcost[j];
            k=j;
        } / * 找到边(losest[k],k)是最小权的边 * /
        tree[p][0]=closest[k];tree[p][1]=k;
        tree[p][2]=min; p++;
        c+=min; / *累计生成树的代价 * /
        lowcost[k]=0; / * 在 U 中加入顶点 k * /
/ *因 k 加入到 U，使 V-U 中，与 K 邻接的顶点到 U 中顶点的代价可能变小 * /
    for(j=0;j<n;j++)
        if(w[k][j]!=0&&w[k][j]<lowcost[j]){
            lowcost[j]=w[k][j]; / * 顶点 j 到 U 的代价更小了 * /
            closest[j]=k;
        }
    }
    return c; / *返回最小代价生成树的代价 * /
}
```

8．7．3 最短路径

在图中求最短路径问题有两个不同的提法：一是求从某个顶点到其他顶点的最短路径。二是求每一对顶点之间的最短路径。

1． 求从某个顶点到其他顶点的最短路径。

Dijkstra 提出了一个按路径长度不减次序产生最短路径的算法。其基本思想是：记指定的某顶点为 v，把图中顶点集合 V 分成两组，以已求出最短路径的顶点集合为第一组，记为 S；其余尚未确定最短路径的顶点集合为第二组。按最短路径长度递增次序逐个地把第二组中顶点移入 S 中，直至从指定顶点出发可以到达的顶点都在 S 中。在这个过程中，需始终保持从 v 到 S 中各顶点的最短路径长度都

不大于从 v 到第二组的任何顶点的最短路径长度。另外，为便于程序处理，需为每个顶点保存一个距离。S 中顶点的距离就是从 V 到此顶点的最短路径长度，第二组中的顶点距离是从V到此顶点的只包括以S中的顶点为中间顶点的那部分还不完整的最短路径长度。用算法描述如下：

[算法]求从顶点 v 到其他顶点的最短路径的 Dijkstra 算法。

```
{ / * 令两维数组 M []]是图的邻接矩阵 * /
    为各顶点设置距离，记 i 顶点 v 的距离为 dist[i];
    让集合 S 只含 v ;
    for(i=0;i<n-1;i++){/ * 以下工作重复 n-1 次 * /
        在 v-s 中选择一个顶点 u，使 dist[u]为最短;
        将顶点 u 加入集合 s 中;
        对于每个属于 v-s 的顶点 w，且 w 与 u 邻接，调整从 v 到 w 距离;
            if(dist[w]>dist[u]+M[u][w])
                dist[w]=dist[u]+M[u][w]l
    }
}
```

以下按上述算法编写的函数。该函数在数组参数 dist 中求得从开始顶点 v 到图中各顶点的最短路径长，在数组参数 pre 中求得在最短路径中各顶点的前驱顶点。

[函数]从求顶点 v 到其他顶点的最短路径 Dijkstra 算法。

```
int shortesPath(int m[][N],int n,int v,int dist[],int pre[])
{    int s[N],i,i,u,min;
    for(i=0;i<n;i++)
        dist[i]=m[v][i]; / * 为各顶点设置距离 * /
        s[i]=0;
        pre[i]=dist[i]<MAX_iNT? V:0;
    }
    s[v]=1; / *让集合 S 只含 v * /
    pre[v]=0;
    for(i=0;i<n-1;i++){
        min=MAX_INT; / * 在 V-S 中选 u，使 dist[u] 为最短 * /
        u=-1;
        for(j=0;j<n;j++)
            if(s[j]==0) / * j 属于 v-s * /
                    if (dist[j]!=0 &&dist[j]<min){
                        min=dist[j];
                        u=j;
                }
        if(u==-1)return 0; / * 不连通的图 * /
        s[u]=1; / *将顶点 u 加入到集合 s 中 * /
```

```
        for(j=0;j<n;j++)
            if(s[j]==0&&m[u][j]<MAX_iNT)
                if(dist[j]>dist[u]+m[u][j])
                        dist[j]=dist[u]+m[u][j];
        }
    return 1;
}
```

2. 求每一对顶点之间的最短路径

这里介绍由 Folyd 提出的求每一对顶点之间的最短路径的方法。设图 G=（V-E）是有 n 个顶点有向图，顶点集合 V={0，1，…，n-1}。图用邻接矩阵 M 表示。Floyd 算法的基本思想是递推地产生一个矩阵序列 A0，A1，A2，…，An，其中 A0，是已知的带权邻接矩阵，Ak(i,j)(0≤i,j<n)表示从顶点 i 到顶点 j 的中间顶点序号不大于 k 的最短路径长度。如果 i 到 j 的路径长度。 如果 i 到 j 的路径没有中间顶点，则对于 0≤k<n,有 Ak(i,j)=A0(i,j)=A0(i,j)=M[i][j].递推地产生 A1，A2，…,An 的过程就是允许逐步越来越多的顶点作为路径中间顶点，直到为全部路径都找到了所有的中间顶点，所有的最短路径也就全部求出，算法就此结束。

设在第 k 次递推之前已求得 Ak-1(i,j)(0≤i,j<n),为求 Ak(i,j)需考虑以下两种情况：

（1） 如果从顶点 i 到顶点 j 的最短路径不经过顶点 k，则由 Ak(i,j)定义知，从 i 到 j 中间的顶点序号不大于 k 的最短路径长度还是 Ak-1(i,j)即 Ak(i,j)=Ak-1(i,j)。

（2） 如果从顶点 i 到顶点 j 的最短路径要经过顶点 k，则这样的一条路径是由 i 到 k 和由 k 到 j 的两条路径的所组成的。由于 Ak-1(i,k)和 Ak-1(k,j)分别表示 i 到 k 和从 k 到 j 的中间顶点序号不大于 k-1 的最短路径长度，若 Ak-1(i,k)+Ak-1(k,j)<Ak-1(i,j)，Ak-1(i,k)+Ak-1(k,j)必定是 i 到 j 的中间顶点序号不大于是 k 的最短路径的长度，则 Ak(i,j)=Ak-1(i,k)+Ak-1(k,j)。
由此得到计算 Ak(i,j)的递推公式如下：

A0(i,j)=M[i][j]

Ak(i,j)=min{Ak-1(i,j),Ak-1(i,k)+Ak-1(k,j),0≤i,j,k<n}.

对于邻接矩阵元素约定如下：

- m[i][i]=0
- 0<m[i][j]<MAX_INT,<i,J>属于 E（G）
- m[i][j]=MAX_INT ，<i,j>不属于 E（G）

下面函数执行后，在数组元素 a[i][j]中存放从顶点 i 到顶点 j 的最短路径的长度。如果 a[i][j]=MAX_INT，则表示 i 到 j 没有路径。数组 path[i][j]存放从 i 到 j 的最短路径的一个中间顶点，若 path[i][j]=-1.则没有中间顶点。细节见以下函数定义。

[函数]用 Floyd 算法求每一对顶点之间的最短路径。

```
int a[N][N];
Void floyd(int m[][N],int n)
{    int i,j,k;
    for(i=0;i<n;i++)
            for(j=0;j<n;j++){
                a[i][j]=m[i][j];
                path[i][j]=-1;
            }
    for(k=0;k<n-1;k++)/ * 递推 n-1 次 * /
            for(i=0;i<n;i++)
                    for(j=0;i<n;j++)
                        if(a[i][k]+a[k][j]<a[i][j]){
                            a[i][j]=a[i][k]+a[k][j];
                                path[i][j]=k;
                    }
   }
```

8．7．4 拓扑排序

设 S 是一个集合，R 是 S 上的一个关系，若 S 中的元素 a,b 有（a,b）∈R，则称 a 是 b 关于 R 的前驱，b 是 a 关于 R 的后继。若对 S 中的元素 a,b,c 有（a,b）∈R 和（b,c）∈R, 则必有(a,c)∈R，就称 R 是 S 上的一个传递关系。若对 S 中的任一元素 a,都有(a,a)∈R，则 R 是 S 上的一个自反关系；若对 S 中的任一元素 a，都不存在(a,a)∈R，则称 R 是 S 上的一个非自反关系。若 S 上的一个关系 R 是传递的并且是非自反的，则称 R 是 S 上的一个半序关系。在一个具有半序关系的 R 的有限集合中，至少有一个元素没有前驱，也至少有一个元素没有后继。

设 R 是集合 S 上的一个半序关系，A=｛a0 ，a1 ，…，a n-1 ｝是 S 中的一个序列，且当(aj,ai)∈R 时，有 i<j，则称 A 是对于 R 的一个拓扑序列。注意在拓扑序列中，若 ai 和 aj 对 R 没有关系，那么 ai 和 aj 在序列 A 是相对位置的前或后不受限制。将一个序列对其重排列使其成为一个拓扑序列的过程称为拓扑排序。

一个大工程有许多项目组，有向图能表示工程项目实行的先后关系。工程中的有些项目对实行先后没有要求,有些项目的实行则存在先后关系,某些项目必须在其他一些项目完成之后才能开始实行。工程项目实行的先后关系可以用一个有向图来表示，工程的项目称为活动，有向图的顶点表示活动，有向边表示活动之间

开始的先后关系。这种有向图称为用顶点表示活动网络，简称 AOV 网络，见图 8.7。

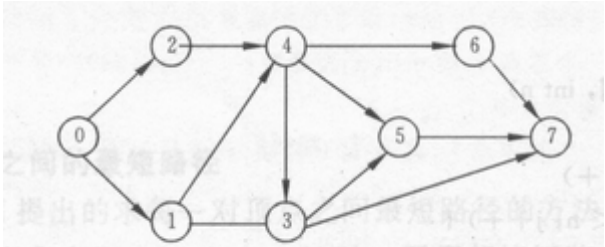


图 8.7 一个 AOV 网络

对 AOV 网络的顶点进行拓扑排序，就是对全部活动排成一个拓扑序列，使得如在 AOV 网络中存在一条弧(i,j)，则活动 i 排在活动 j 之前。对图 8.7 的有向图的顶点进行拓扑排序，可以得到多个不同的拓扑序列，如：

02143567
02143657
02143657
01243567

很显然，一个存在回路的有向图的顶点是不能排成拓扑序列的，因为有回路有向图的边所体现的先后关系不是非自反的。任何没有回路的有向图，其顶点可以排成一个拓扑序列。下面是对 AOV 网络进行拓扑排序的算法：

【算法】AOV 网络拓扑排序。

```
{     置拓扑排序的结果序列为空;
        do {
                在网络中任选一个没有前驱的顶点，并将它存入序列;
                从网络中删除选中的顶点和自该顶点出发的有向边;
        } while(!网络已没有顶点   网络中的顶点都有前驱顶点));
    /·如果网络已没有顶点，则队列中的顶点序列即为一个拓扑序列·/
}
```

设采用邻接表示 AOV 网络，为了便于判定一个顶点是否有前驱，引入数组存储各顶点的入度，则那些入度为 0 的顶点就是没有前驱的顶点。又为了便于程序寻找入度为 0 的顶点，程序将所有入度为 0 的顶点将它们的下标栈式链接。细节见以下函数。

【函数】AOV 网络拓扑排序。

```
int topol_sort(Gragp g,int n,int queue[])
{     int top.count,i,j;
        edgeNode *p;
```

```
        int inCount,[N];
        for(i=0;i<n,i++)inCount[i]=0;
        for(i=0;i<n;i++)
                for(p=g[i];p!=UNLL;p=p->next) {
                j=p->vno;
                inCount[j]++;
        }
        top=-1;
        for(i=0;i<n;i++)
            if(inCount[i]==0) {
                inCount[i]=top;
                top=i;
        }
        count=0;
        while(top>=0) {
            queue[count ++]=top;
            p=g[top];
            for(;p!=NULL;p=p->next) {
                j=p->vno;
                if(--inCount[j]==0) {
                inCount[j]=top;
                top=j;
                }
            }
        }
    }
    return count ==n;/*返回 1，拓扑排序成功·/
}
```

8.7.5 关链路径

利用 AOV 网络，对其进行拓扑排序能对工程中的活动的先后顺序作出安排。但一个活动的完成总需要一定的时间，为了能估算某个活动的开始时间，找出那些最影响工程完成时间的活动，边上的权表示完成该活动所需的时间，这种用边表示活动的网络称为 AOE 网络。

例如，图 8.8 表示一个具有 12 个活动的某个工程的 AOE 网络。图中有 8 个顶点，分别表示事件 0-7，其中 0 表示工程开始状态，7 表示工程结束状态，边上的权表示完成该活动所需的时间。

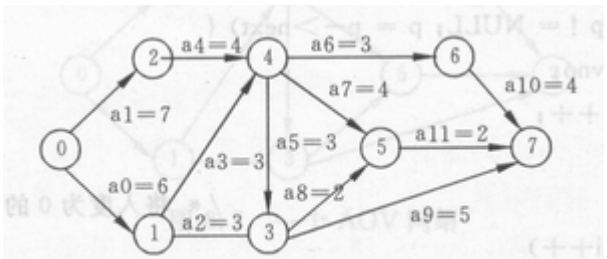


图 8.8 一个 AOE 网络

在实际中，AOE 网络没有回路，存在惟一的入度为 0 的开始顶点，和存在惟一的出度为 0 的结束顶点。在 AOE 网络上，要研究的问题是完成整个工程至少需要多少时间，哪些活动是影响工程进度的关键。

因 AOE 网络中的某些活动可以并行地进行，所以完成工程的最少时间是从开始顶点到结束顶点的最长路径长度，称从开始顶点到结束顶点的最长路径为关健路径，关键路径上的活动为关键活动。

为了找出给定的 AOE 网络的关键活动，从而找出关键路径，先定义几个重要的量：

(1)数组 est[i],est[i]存取事件 v1 能够发生的最早时间。est[i]是从开始顶点 0 到顶点 i 的最长路径的长度。

(2)数组 elt[i],elt[i]存储事件 v1 最迟发生时间，这是保证结束顶点在 est[n-1 时刻发生的前提下，事件 v1 允许发生的最迟时间。elt[i]等于 est[n-1]减去顶点 i 到顶点 n-1 的最长路径长度。
(3)数组 ast[i],ast[i]存储活动 ai(j,k)ai 的最早开始时间 ast[i]等于从开始顶点 0 精通表示事件 vi 的顶点 j 的最长路径长度，即 ast[i]=elt[i]。

(4)数组 ast[i],ast[i]存储活动 ai 允许的最迟开始时间 alt[i]等于 elt[k]减去活动 ai 所需时间.
这样，ast[i]=alt[i]的活动就是关键活动，因活动中 ai 如拖延时间，则整个工程也要拖延。而 alt[i]-ast[i]是活动 ai 最大可拖延的时间，那些 alt[i]-ast[i]>0 的活动就不是关键活动。

由以上约定，计算出这些量。就能求出关键活动和关键路径，算法如下：

【算法】计算 AOE 网络的关键路径。

```
{     输入 AOE 网络有向边，建立图的存储结构;
      if(对图作拓扑排序!=网的顶点数) {
              输出网有回路字样;
              exit(1)/*结束程序*/
      }
      est[0]=0;
```

从顶点出发，按顶点的拓扑顺序求其余各顶点的最早发生时间：
est[k]=max{est[j]+边<j,k>上的权,其中 j 是顶点 k 的前驱顶点};

/*上式表示在顶点 k 的所有前驱顶点中求最大值*/
est[n-1]=est[n-1];
从结束顶点 N-1 出，按逆拓扑顺序求其余各顶点的最迟发生时间：
elt[k]=min{elt[j]-边<k,j>上的权，其中，J 是顶点 K 的后继顶点};
/*上式在顶点 K 的所有后继顶点中求最小值·/
由 est[]和 elt[]，求每条弧的最早开始时间 ast[]和最迟开始时间 alt[];顺序考察 est[]和 alt[]，若某条弧 i 满足条件 ast[i]==alt[i]；则闭弧 i 是关键活动。
}

习惯计算 ast[i]的值与拓扑排序一起进行，以省去搜索顶点 K 的所有前驱站点，否则需采用有向图的逆邻接表示形式。限于篇幅，找 AOE 网络关键活动的完整程序留给读者作为练习。

【例 8.17】某个城市有 N 个车站，并有 M 条公交路连接这些车站。设这些公交车都是单向的，这 N 个车站被顺序编号为 0 至 N-1。现要求编制程序，输入城市的车站数 N、公交线路数 M 及各共交线路的站号。若乘坐问题所给的公交车，求从站 0 至站 N-1 的最少上车次数。

程序近输入信息构建一张有向图 G，有向图的顶点是车站，若有某条公交线路径 i 站能到达 J 站，就在顶点 I 到顶点 J 之间设置一条权为 1 的有向边(I，J)。从站点 X 至站点 Y 的最少上车次数便对应图 G 从点 X 至点 Y 的最短路径长度。

【例 8.17 程序】

```
#include<stdio.h>
#include<stdib.h>
int*buildG(int*np)
 {    int n,m,sc,i,j,k;char ch ;int*s,*g;
        printf(" 输入车站个数和分交线路条数：" )
        scanf(" %d%d",&n,&m);
        g=(int*)malloc(sizeof(int)*n*n); /*用一维数组实现两维数组的要求*/
        for(i=0;i<n*n;o-i+ +)
            g[i]=0 /*预置各站点互不可达*/
        s=(int*)malloc(sizeof(int)*n);
        for(i=0;i<m;i++) { /*输入各公交线路径历的站点*/
            printf(" 输入第%d 条公交车线路的各站编号(0<=编号<=%d):\n,i+1,n-1" );
            printf(" 要求在线路的最后一个站号输入后紧接输入回车。\n" );
            sc=0; /*用一维数组实现两维数组要求*/
```

```
            ch=' ' ; /*预置各站点互不可达*/
        while(ch!=' \n' ) /*输入各公交线路径历的站点*/
            scanf(" %d%c" ,&s[sc++],&ch);
        for(j=0;j<sc;j++ ) /*检查当前线路各站点的合理性*/
            if(s[j])<0   s[j]>n)break;
        if(j<sc){ /*有不合理的站点号，重新输入*/
            printf(" 输入错误，该线路重新输入" \n),i--;contonue;
            }
            for(k=1;k<sc;k++)
                for(j=0;j<k,j++)/*当前线路中，设置前面各站至当前站的有
向边*/
                    *(g+s[j]*n+s[k])=1;/*可从 s[j]站到达 s[k]站*/
            }
        free(s);
        *np=n;
        rerturn g;
}
int minLen(int *g,int n)
 {    int j.k,*dist;
        if(n==0)return 0;
        dist=(int*),malloc(sizeof(int)*n)/*存储到各站上车次数*/
        for(j=0;j<n;j++)/*以从0站可直达站的上车次数(0/1)为dist 设置初
值*/
            dist[j]=g[j];
        dist[0]=1;/*若 dist[i]<0,, 则 i 站的上车次数已确定为 dist[i]*/
        do {/*每循环一次计算到一个站的换车次数*/
            for(k=-1,j=0;j<n;j++) /*找下一个最少上车次数的站*/
                if(dist[j]>0&&(d==-1 dist[j]<dist[k]))
                    k=j;
                if(k<0 k==n-1)
                    break;/*找不到未处理的车站，或确定了 n-1 站的上车次
数*/
            dist[k]=-dist[k];/*设置 k 站已求得上车次数的标记*/
            for(j=1;j<n;j++)
/*因到 k 站已确定上车次数，修正还未确定并能从 k 站可达的所有站的上车次数
*/
                if(dist[j]>=0&&*(g+k*n+j)&&
                        (dist[j]>==0 - dost[k]+1<dist[j]))
                    dist[j]=-dist[k]+1
        }While(1):
        j=dist(n-1)
        free(dist);
        return<0 ?-1:j;
}
```

```
void main()
 {
    int*g,t,n;
    g=buildG(&n);
    if((minLen(g,n))<0)printf(" 无解!\n" )
    else printf(" 从 0 号站到%d 站需上车%d 次\n" ,n-1,t)
    free(g);

}
```

8.8 排序

对于有 N 个结点的线性表(e0 ，e1 …e n-1)，按照结点某些数据项的值按递增或递减的次序，重新排列线性表结点的过程，不为排序，排序时参照的数据项称为排一，通常选择结点的键值作为排序码。若线性表中排序码相等的结点经某种排序方法进行排序后，仍能保持它们在排序之前的相对次序，称这种排序方法是稳定的；否则，这种排序方法是不稳定的。

在排序过程中，线性表的全部结点都在内存，并内存中调整它们在线性表中的存储顺序，为内排序。排序过程中，线性表只是部分结点被调入内存，并借助内存调整结点在外存中的的排序方法称为外排序。

为了叙述方便，在讨论序方法中，假定线性表的结点只含键值信息待排序的 N 个结点的线性表记为(e0 ，e1 …e n-1)，并要求按递增顺序排序。

1.选择排序

选择排序的基本思想是反复从还未排好序的那部分线性表中选出键值最小的结点，并按从线性表中选出的顺序排列结点，重新组成线性表。直至未排序的那部分为空，则重新形成的线
性表是一个有序的线性表。

(1)顺序存储线性表的选择排序

若线性表用顺序存储实现，为存放选出的结点并重新组成线性表，选择排序算法可具体为：选出键值最小的结点与线性表的第 1 个结点交换，选出键值次最小的结点与线性表的第 2 个结点交换，依此类推，直至选出键值次大的结点与线性表的最末第 2 个结点交换。至此，键值最大的结点成为线性表的最末点。即依次对 i=0,1,2,…n-2 分别执行如下的选择交换步骤：

在结点序列 e[i],e[i+1],…,e[n-1]中选择键值最小的结点 e[k];

当 k 不等于 i 时,将 e[i]与 e[k]交换.

经上述 n-1 次选择和交换,最终完成顺序线性表排序。写成函数如下：

```
void ss sort (
    int e[]. /*存储线性表的数组*/
    int n) /*线性表的结点个数*/
{    int i,j.k,t;
       for(i=0;i<n-1;i++) { /*控制 n-1 遍的选择步骤*/
    /*  在 e[i].e[i+1]…e[n-1]中选键值最小的结点 e[k]*/
       for(k=i;y=i+i;j<n;;j++)
              if(e[k]>e[j])k=j;
       if(k!=i){ /*e[i]与 e[k]
              t=e[i];e[i]=e[k];e[k]=t;
       }
    }
}
```

n 个结点的线性表采用选择排序，当外循环 I=0 时，内循环要做 n-1 比较；当听循环 I=1 时，内循环要做 N-2 比较；依此类推，当个循环 i=n-2 时，内循环要做 1 次比较。因此，总共比较次数为 n(n-1)/2。因为相等的两元素，相对在前的可能交换到更后面，所有选择排序是不稳定的。

(2)链接存储线性表的选择排序

对于链接存储的线性表，选择排序过程中，依次选出的键值从小到大的结点可顺序组绢成有序链表,即每次选出的键值最小的结点接在已部分完成的有序链表的末尾。写成函数如下：

【函数】链接存储线性表的选择排序。

```
struct node*ls_sort(strict node*h)
{    struct noee *q,*t,*s,*tail,*u,*v;
       s=NULL;/*置有序链表头指针为空*/
       while(h!=NULL){/*在剩余的链表中找键值最小的结点*/
           for(t=u=h;u->link!=NULL;u=u->link)
               if(u->link->data<t->data{
                   v=u;/*保留键值更小的结点的前驱结点指针*/
                   t=u->link;/*找到一个键值更小的结点*/
               }
               if(t==h)
                   h=h->link; /*键值最小结点是链表的首结点*/
               else v->link=t->link;/*键值最小结点脱钩*/
           if(s==NULL)
               tail=s=t; /*第一次找到键值最小结点*/
           else tail=tail->link=t;
       }
```

```
        if(s!=NULL)tail->link=NULL; /*让有序链表到此结束*/
        resurn s;/*返回有序链表首结点指针*/
}
```

2.直接插入排序

直接插入排序的基本思想是假设线性表的前面 i 个结点序列 e ,e1 ,e2 是已排序的。对结点 e 在这有序结占序列中找插入位置，并将 e 插入，而使 i+1 个结点序列 e0 ,e1,e2bm 也变成排序的。依次对 i=1,2…n-1 分别执行这样的插入步骤，最终实现线性表的排序。
(

1)顺序存储线性表直接插入排序

若线性表是顺序存储的，为寻找 e[i]的插入位置物完成插入工作，可采用以下步骤：先将 e[i]暂存于某变量 t，空出存储 e[i]的空间，然后将 t 依次与 e[i-1]e[i-2]…进行比较，将比 t 大的结点依次右移一个位置，直到发现某个 j(0<=j<i),有 e[j]<=t 成立，则把 T 存放入 e[j+i]
.如果这样的 j 不存在，那么比较过程使 e[i-1],e[i-2]…e[0]都依次右移一个位置，此时将 t 存入 e[0].经如此 n-1 遍的比较、移动、插入步骤之后，线性表就排好了序。按上述想法有以下函数定义：

【函数】顺序存储线性表的直接插入排序。

```
void si sort
    int e[] /*存储线性表的数组*/
    int n) /*线性表的结点个数*/
 {    int i,j,t ;
        for (i=1;i<n;i++){/*控制 e[i],……,e[n-1]的比较插入步骤*/
/* 找结点 e[i]的插入位置*/
        for(t=e[i],j=i-1,j>=0&&t<e[j];j--)
            e[j+i]=e[j];
            e[j+1]=t /*插入*/
        }
}
```

对 n 个结点的线性表采用直接插入排序，当线性表已是从小到大排序时，内循环每执行一次，只需作 1 次比较，整个排序只进行 n-1 次比较。当线性表已是从大到小排序时,对外循环 i,内循环要进行 i 次比较。整个排序过程需进行 n(n-1)/2 比较。因此，对 n 个结点的线性表采用直接插入排序时，比较次数最少为(n-1)次，最多为 n(n-1)/2。直接插入排序有原来顺序好排序效率也高的特点，而且直接插入排序是稳定的。

(2)链接存储线性表的直接插入排序

对于链接存储的线性表，一步插入排序过程包括为未排序的结点找到插入处。上述过程直至不再有还未排序的结点，完成链接线性表的排序。细节见以下定义。

【函数】链接存储线性表的直接插入排序

```
Struct node *li_sort(struct node *h)
{    struct node *t, *s, *u, *v ;
       S=h->link ; /* s 为还未排序结点序列的首结点指针  */
       h->link = NULL ; /* 只含一个结点的链表为有序链表  */
       while (s ! =NULL ) {
           /* 在有序链表中找插入位置  */
         for ( t = s , v = h ; v ! = NULL & & v - > data < t - > data ;
                  u = v , v = v - > link ) ;
                  s = s - > link ; /* 预先为考虑下一个未排序结点做准
备  */
                  if ( v = = h ) h = t ; /* 插在首结点之前  */
                  else u - > link = t ;
                  t - > link = v ; /* 完成插入*/
    }
    return h ;
}
```

3.冒泡排序

设线性表的结点依次从上而下纵向排列。冒泡排序的基本思想是对当前还未排好序的范围内的全部结点，自上而下对相邻的两个结点依次进行比较和调整，让键值较大的结点往下沉，键值较小的结点往上冒。即，每当两相邻结点比较后发现它们的排列顺序与排序要求相反时，就将它们互换。开始时，排序范围是从 e0 至 en-1，以后的排序范围由前一遍的扫视结果确定：当自上而下在当前排序范围内的结点执行一遍比较之后，假设最后往下沉的结点是 ej，ej 下沉到 ej+1，ej+1 以下的结点比较发现它们都不再需要交换。这说明 ej+1 以下的结点是已排好序的，则下一遍的排序范围可立即缩减为从 e0 至 ej。整个排序过程最多执行遍 n - 1。若某一遍的比较没有结点交换，所有相邻结点的排列顺序都与排序要求一致，则线性表是已排序的，就不需要再进行一遍比较和调整。

（1） 顺序存储线性表的冒泡排序

若顺序存储线性表用数组实现。初始时，排序范围的终点 h 是 n - 1，排序过程在排序范围的终点 h 大于 0 情况下循环，为了记录一遍扫视最后下沉的结点位置 p，在每遍扫视前先将 p 预置初值 0，以后有结点下沉时，用下沉结点的下标更新 p。一遍扫视结束，值 p 就可作为下一遍扫视范围的终点 h。细节见以下函数定义。

【函数】 顺序存储线性表的冒泡排序。

```
Void sb - sort (
    Int e [] , /*  存储线性表的数组  */
    Int n ) /*  线性表的结点个数  */
{ int j , p , h , t ;
    for ( h = n - 1 ; h > 0 ; h = p ) { /*  循环至没有比较范围  */
    for ( p = j = 0 ; j < h ; j + + )
        if ( e [ j ] > e [ j + 1 ] ) { /* 比较后，发现要交换*/
        t = e [ j ] ;
        e [ j ] = e [ j + 1 ] ;
        e [ j + 1 ] = t ;
        p = j ;
        }
    }
}
```

n 个结点的线性表采用冒泡排序，冒泡排序的外循环最多执行 n－1 遍，第一遍最多执行 n－1 次比较；第二遍最多执行 n－2 次比较；依此类推，第 n－1 遍最多执行 1 次比较。因此，整个排序过程最多执行 n（n－1）/2 比较。因键值相等的结点不交换，冒泡排序是稳定的。

（2） 链接存储的线性表的冒泡排序

对于链接存储的线性表，一遍扫视范围不再有结点的下标可供参照，每遍扫视都从链表的首结点开始，第一遍直至末结点，以后各遍扫视直至上一遍扫视最后下沉的前驱结点。为了便于控制扫视范围，可用扫视范围中最后一个结点的指针值作为扫视范围结束的依据。第一遍直至线性表的末结点的后继指针值空，以后各遍扫视直至指针为上一遍最后下沉的那个结点的指针。另外，当两相邻的结点需要作下沉上冒的交换时，需要更改相邻结点中前一个结点的前驱结点的后继指针。为此在排序前，先在链表的首结点之前插入一个辅助结点，待排序结束后，再释放该结点。

【函数】 链接存储线性表的冒泡排序

```
struct node * lb_sort (struct node * h )
{ struct node * endpt , * u , * v , * p ;
    u = ( struct node * ) malloc ( sizwof (sturct node ) ) ;
    u - >link = h ;
    h = u ;
    for ( endpt = NULL ; endpt ! = h endpt = p )
        for ( p = u = h ; u - > link ->link ! = endpt ; u = u - > link )
            if (u - > link - > data > u - > link - > link - data ) {   /*
需要交换  */
                v = u - > link - > link ; u - > link - > link = v - > link ;
                v - > link = u - > link ; u - > link = v ;
                p = u - > link - > link ;
```

```
        }
    u = h ; h = h - > link ; free ( u );
    return h ;
}
```

4. 希尔排序

在直接插入法中，每插入一个结点，使有序序列只增加 1 个结点，并且对插入下一个结点没有提供任何帮助。考虑到直接插入法有原来顺序好排序效率就高的特点，如把直接插入方法分成插入步长由大到小不同的若干趟来进行，一开始步长较大，相当于把序列分成几个子表，对每个子表来说，因其结点少，直接插入排序效率会很高。以后各趟逐步减小步长，子表的结点也越来越多，但子表中的结点已经过前一趟的在步长的直接插入排序，有相当多的结点已排好了序，这样，后一趟的插入排序就能充分利用前一趟的排序结果。因前一趟的排序使序列的顺序性变好，而后一趟的排序效率就会得到提高。最后一趟的步长为 1，就是对已几乎有序的整修序列作一次直接插入排序。

上述排序方法称为希尔(Shell)排序。希尔排序循环地进行 t 趟(t < tranc (log I (n + 1)))插入排序，每趟步长递减，但最后一次必须为 1。具体取什么样的步长序列尚无最好的推荐结论。希尔排序对直接插入排序来说，在效率方面有很大的改进。下面的函数是一个希尔排序算法的实现，初次取序列的一半为步长，以后每次减半，直至步长为 1。

【函数】 希尔排序函数

```
void shell (int e[],int n)
{ int j , k , h , y ;
   for(h=n/2;h>0;h=h/2)              /* 共扫视 log 2 ( n +1 ) 趟*/
   for(j=h;j<n;j++){                 /* 同一子表内作直接插入排序
                                     * /
                                     /* 让出当前考察元素的存储位
      y=e[j] ;                       置 */
   for(k=j-h;k>=0&&y<e[k];k-=h)
      e[k+h]=e[k];                   / * 找插入位置，并移动  */
      e[k+h]=y;                      / * 插入 * /
   }
}
```

5. 堆排序

堆排序是一种树形选择排序，是对直接选择排序的有效改进。

堆排序来源于一种称为比赛树的排序方法。用比赛树进行排序的方法是：先对 n 个结点的键值进行两两比较，再对其中 n/2 个较小者之间作两两比较，依此类推，直至选出键值最小的结点。这个过程可用一棵有 2n-1 个结点的丰满二叉树来表

示，二叉树的叶子结点是待排序的结点序列，二叉树的非叶子结点是层层比较产生的结点。

上述一轮的 n-1 次比较，在根结点选出最小的键值结点。然后将已选出的结点改为无穷大，并从该结点开始和其兄弟结点比较，修改从叶结点到根结点路径上的各结点值，则根结点为次最小者。依此类推，选出从小到大的各结点。总之，除第一个最小者需比较 n - 1 次外，选其他任一结点都只需从叶结点到根结点路径上那些结点的比较，其比较次数与二叉树的高度相对应，比较次数为 0(log2n)。完成排序的总比较次数为 0(nlog2n)。

堆排序是对比赛树排序方法的改进。一个堆是这样一棵顺序存储的二叉树，它的所有的父结点(e [i])的键值均不小于它的左子结点(e [2*i+1])和右子结点(e [2*i+2])的键值。初始时，若把待排序序列的个结点看作是一棵顺序存储的二叉树，调整它们的存储顺序，使之成为一个堆，这里堆的根结点键值是最大者。然后将要结点与堆的最后一个结点交换，并对少了一个结点后的结点重新作调整，使之再次成为堆。这样，在根结点得到结点序列键值次最大者。再次将堆的根结点与堆的最后一个结点交换，并重新使又少了一个结点后的序列调整成为堆。依此类推，直至只有两个结点的堆，并对它们作交换，最后得到有序的个结点序列。

堆排序的关键步骤有两个，一是如何建立初始堆。二是当堆的根结点与堆的最后一个结点交换后，如何对少了一个结后的结点序列作调整，使之重新成为堆。

对于交换后重新建堆来说，交换只能是二叉树根结点可能会破坏堆的性质，为此将它与它的两个子结点比较，如根结点比它的两个子结点都大，则已经是堆；否则根结点与其中较大的子结点交换，先让根结点满足堆的性质。可能因交换，使以交换后的结点为根的子树不再满足堆的性质，则重复向下调整。当调整使新的更小的树依旧满足堆的性质时，重新建堆过程结束；当交换使新的更小的子树不再满足堆的性质时，继续按上述方法调整被破坏的更小子树。最坏情况是直至调整到中结点才结束。这种自上而下调整建堆的过程称为结点向下渗透。

为建立初始堆，可从已是堆的两个子序列上面加上它们的根结点，并作必要的调整使之成为更大的堆。加上根结点后，可能不满足堆的性质时，则又可用前述的渗透方法，使之成为堆。首先二叉树的叶结点分别都是堆，然后从第 n/2 结点开始，依次将第 n/2 结点，第 n/2-1 结点，…. ，第 1 结点逐一加到它们的子结点上，直到建成一个完全的堆。

设待排序的结点键值序列为 e [ ] = { 93,35,29,45,48,82,76,17}，按上述思想形成如图所示的初始堆。

堆排序有两个函数组成，一是实现渗透的函数，另一个是反复调用渗透函数实现堆排序的函数。细节见以下函数定义。

【函数】 渗透函数

```
void sift (
    int e [ ] , /* 存储结点序列 */
```

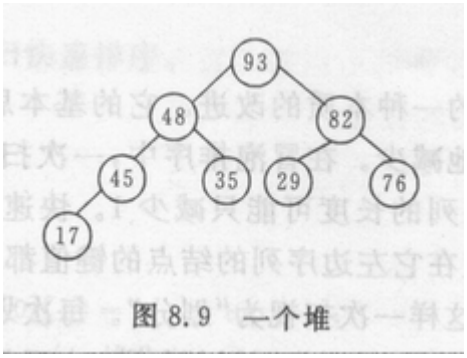


图 8.9 一个堆

```
    int n,  /*序列的结点个数 */
    int s)  /*对 e[e]作渗透，其中 e[s]的左右子树是堆 */
{    int t, k, j;
    t=e[s];
    k=s; j=2*k+1; /*e[k]是 e[ j ]和 e[j+1]的父结点 */
    while (j< n) { /*未渗透到叶结点 */
        if (j< n-1 && e[ j ]<e[j+1])
            j++; /*e[j+1]是 e[k]较大的子结点 */
        if (t< e[ j ]) { /*改连续渗透交换为循环传送 */
            e[k]=e[ j ];
            k=j;
            j=2*k + 1;
        } else break; /*已是一个堆 */
    }
    e[k] = t;
}
```

【函数】 堆排序

```
void heapsort (int e [ ] , int n )
{ int i, k, t ;
    for i ( i=n/2-1 ; i >=0 ; i--)
        sift ( e, n, i) ; /* 建初始堆 */
    for ( k = n-1 ; k > = 1 ; k -- ) {
        t = e [ 0 ] ; /* 根结点与堆的末结点交换*/
        e [ 0 ] = e [ k ] ;
        e [ k ] = t ;
        sift ( e , k , 0 ) ; /* 重新建堆 */
    }
}
```

6. 快速排序

快速排序是对冒泡排序的一种本质的改进。它的基本思想是通过一趟扫视后，使待排序序列的长度能较大幅度地减少。在冒泡排序中，一次扫视只能确保最大键值的结点移到了正确位置，而待排序序列的长度可能只减少了 1。快速排序通过一趟扫视使某个结点移到中间的正确位置，并使在它左边序列的结点的键值都比它的小，而它右边序列的结点的键值都不比它的小。称这样一次扫视为"划分"，直至新的子序列的长序列变成两个新的较小子序列，对这两个小子序列分别作同样的划分，直至新的子序列的长度为 1 时才不再划分。当所有的子序列的长度都为 1 时，序列已是排好序的了。

如果每次划分都能得到两个几乎相等的子序列，这种理想情况，排序是最快的。对各种情况，划分次数的量级是 0(log2n)，整个排序的时间复杂度为 0(nlog2n)。相反，如果对 m 个结点的一次划分得到这样两个子序列，其中一个结点数为 m-1，另一个子序列为空，则这是最坏的情况，每次划分仅使待排序序列的长度减少了 1。这种情况的时间复杂度为 0(n2)。

如上所述，快速排序函数需要一个划分函数，快速排序函数是控制对划分得到的子序列的再划分。这有两种可选办法，一种是用递归实现。另一种是引入一个栈，将划分产生的子序列的起讫位置记录在栈中，栈的初态是原告要排序序列的起讫位置。函数反复从栈中取出子序列的起讫位置，并对序列作划分，并将新产生的长度大于 1 的子序列的起讫位置进栈，直至栈为空，结束快速排序过程。以下是快速排序的函数。

【函数】 采用递归方法的快速排序

```
void r_ruick int e [ ] , int low, int high )
{ int i, j , t ;
    if ( low < high ) { / * 对至以为基准作划分 * /
        i = low ; j = high ; t = e [ low ] ;
        while ( i< j ) {
            while (i< j && e [ j ] > t ) j - - ;
            if (i< j ) e (i++) = e [ j ] ;
            while (i< j && e [i] < = t ) i - - ;
            if (i< j ) e (j -- ) = e [ i ] ;
        }
        e [i ] = t ;
        r _ ruick ( e , low , i-1) ; /* 递归，对左子序列作划分* /
        r _ ruick ( e , i+1 , high) ; /* 递归，对右子序列作划分* /
     }
}
```

【函数】 采用栈的非递归快速排序。

```
# define N 40
void s_quick ( int e [ ] , int n )
{ int i , j , low ,high , t ;
    int stack[N][2],top;
    stack[0][0]=0;stack[0][1]=n-1;top=1;
    while(top>0){ / * 栈非空 * /
        top--;
        low=i=stack[top][0];
        high=j=stack[top][0];
        t=e[low];
        while(i<j){ / * 对到为基准作划分 * /
            while(i<j&&e[j]>t)j--;
            if (i<j)E[I++]=e[j];
            while(i<j&&E[I]<=T)I++;
            if(i<j)e[j--]=e[i];
        }
        e[i]=t;
        if(i-1>low){ / * 左边子序列起止位置进栈 * /
                stack[top][0]=low;stack[top][1]=i-1;
                top++;
        }
        if(high>i+1){ / * 右边子序列起止位置进栈 * /
            stack[top][0]=i+1;stack[top][1]=high;
            top++;
        }
    }
}
```

7. 合并排序

合并排序是将两个或两个以上的有序子表合并成一个新的有序表。对于两个有序子表合并成一个有序表的两路合并排序来说，初始时，把含 n 个结点的待排序序列看作有 n 个长度都为 1 的有序子表所组成，将它们依次两两合并得到长度为 2 的若干有序子表，再对它们作两两合并.直到得到长度为 n 的有序表，排序即告完成。

两路合并排序需要和待排序序列一样多的辅助空间，下面的函数 merge (e,a,s,m,n)实现将子序列 e[s],e[s+1],…e[m]和子序列 e[m+1],e[m+2],…e[n]合并到 a[s],a[s+1],…,a[n]中，实现将两个相邻的有序段合并成一个有序段。

【函数】 两个相邻有序段的合并。

```
Void merge_step (int e[ ] , int a[ ] , int s ,int m, int n )
{ int i,j,k;
    k=i=s;j=m+1;
    while(i<=m&&j<=n) / * 当两个有序段都未结束时循环 * /
```

```
    if e[i]<=e[j])a[k++]=e[i++]; / * 取其中小的元素复制 * /
    else a[k++]=e[j++];
    while (i<=m)a[k++]=e[i++]; / * 复制还未合并完的剩余部分 * /
    while (j<=n)a[k++]=e[ij++]; / * 复制还未合并完的剩余部分 * /
}
```

有了 merge_step ( )函数，就可写出合并排序中一趟完整合并，实现将 e[ ]中的每 len 个结点有序的 n 个结点两两合并到，使 a[ ]中的结点每 2 *len 个结点有序。

【函数】 完成一趟完整的合并。

```
Void merge_pass ( int e [ ] , int a [ ] , int n , int len )
{ int f_s,s_end;
    f_s=0;
    while (f_s+len<n){ / * 还至少有两个有序段 * /
        s_end=f_s+2*len-1;
        if (s_end>=n)s_end=n-1; / * 最后一段可能不足个结点* /
        merge_step(e,a,f_s+len-1,s_end); / * 相邻有序段合并 * /
        f_s=s_end+1; / * 下一对有序段中左段的开始下标 * /
    }
    if (f_s<n) / * 当还剩一个有序段时，将其从复制到 * /
    for (;f_s<n;f_s++)
    a[f_s]=e[f_s];
}
```

最后是两路合并函数，该函数主要控制有序段的长度，每合并一趟，有序段长加倍，并交替地从 e[ ]合并到 a[ ]，又从 a[ ]合并到 e[ ]，直到只有一个有序段为止。

【函数】两路合并排序。

```
Void merge (int e[ ],int n)
{ int *p=(int *) malloc(sizeof(int)*n);
    int * p=1,f=0;
    while(len<n){ / * 交替地在和之间来回合并 * /
        if(f==0)merge_pass(e,p,n,len);
        else merge_pass(p,e,n,len);
        len=2; /* 一趟合并后，有序段结点数加倍*/
        f=1-f; /* 控制交替合并*/
    }
    if(f)
            for(f=0;f<n;f++) e[f] = p[f];
    free(p);
}
```

8、外排序

外排序就是对大型文件的排序，待排序的记录存储在外存中，在排序过程中，内存只存储文件的一部分记录，整个排序过程需进行多次的内外存间的交换。

一般来说，外排序分两步进行。第一步，按内存可用空间的大小，将外存的记录分批依次读入内存，并利用某种内排序方法进行排序后输出到另一文件，在新文件中形成许多有序的记录段，称为归并段。第二步，对第一步形成的归并段用某种归并方法进行一趟趟地归并，使文件中的有序段越来越长，直到整个文件成为一个有序段为止。这里介绍其中常用的多路（k 路）平衡归并方法。

k（k>=2）路平衡归并是指文件经外排序的第一个步骤后，已是由若干初始归并组成的文件，反复按每 k 个归并段合并成一个有序段，将一个文件上的记录归并到另一个文件。重复这一过程，直到某趟归并后，文件上的记录成为单个有序段。

由于这趟合并中，文件中的归并段不一定会正好是 k 的倍数，所以可能因只剩下一个归并段，这个归并段暂不能合并；或可能一趟合并过程中的最后一次归并不足 k 个归并段。

K 路归并可采用比赛树的方法来进行合并，其合并过程如下：首先用参加合并的 k 个有序段的第一个记录构造出一棵初始比赛树。具体构造方法是，首先用这 k 个记录作为比赛树的叶结点，接着对相邻的两个结点进行比较，把键值小的记录（优胜者）作为这两个结点的父结点，按此方法自下而上一层层地产生比赛树的结点。通常比赛树的非叶子结点只存放记录的键值和指向记录的指针。当比赛树构造好后，其根结点就是全胜者，是当时 k 个记录中的键值最小者。把根结点所代表的记录输出，并让输出记录所在的归并段的下一个记录参加比赛，即自该记录所在的叶结点出发自下而上与兄弟结点比较，优胜者结点胜出，并让它与上一层的兄弟结点比较，直至根结点，再次得到一个全胜结点输出。如此重复直到 k 个归并段上的全部记录都输出为止，完成一组 k 个归并段合成一个有序段。对于 k=8，图 8.10 是一个初始化比赛树，图 8.11 是输出一个记录后，重新调整后的比赛树。

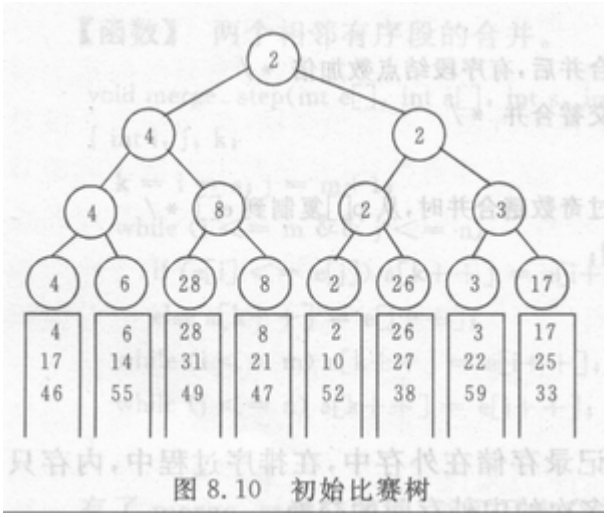


图 8.10 初始比赛树

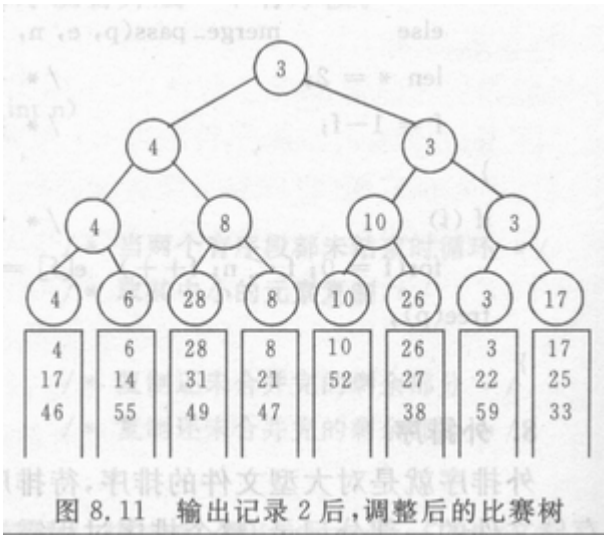


图 8.11 输出记录 2 后，调整后的比赛树

在上述方法中，树中结点存储优胜者的信息，所以也称为胜者树方法。为程序设计简便，宜将上述方法稍作改变。在建树和调整过程中，让胜者继续参加上一层的比赛，而结点记录失败者的信息，这样改变后，称为败者树方法。这种改变的优点是在调整过程中，当优胜者参加上一层比赛时，到了父结点就立即知道原先在此比较的失败者，而直接与原失败者比较，并再次记录新的失败者和让优胜者出去参加更上一层的比赛。而在胜者树中，向上调整需与兄弟结点比赛，结点需要保存兄弟结点的位置信息。所以败者树更便于编程。

为了简便，设记录只有一个排序码；败者树用数组 lost[ ] 顺序存储，其中的值为归并段序号，令 lost[1]是树根结点，lost[0]是根的父结点，它存储每次选出的归并段号，输出的将是该归并段的当前记录；各归并段参与归并选择的排序码存于数组 s[ ]。对于 k=8，图 8.12 是一个初始败者树，图 8.13 是输出一个记录后，重新调整后的败者树。这里给出 k 路归并败者树方法的调整和一次 k 路归并的函数如下：

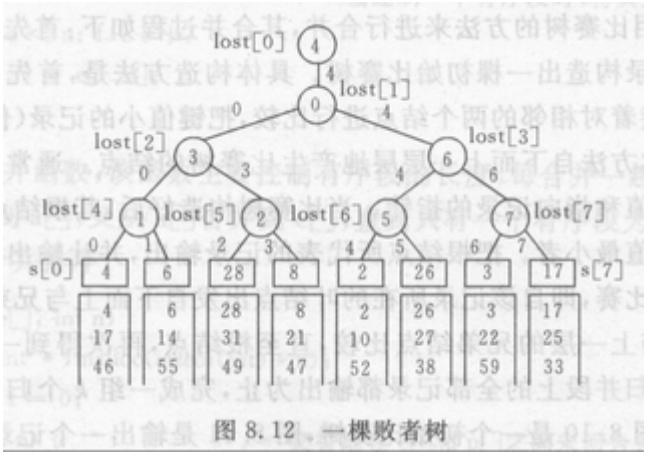


图 8.12 一棵败者树

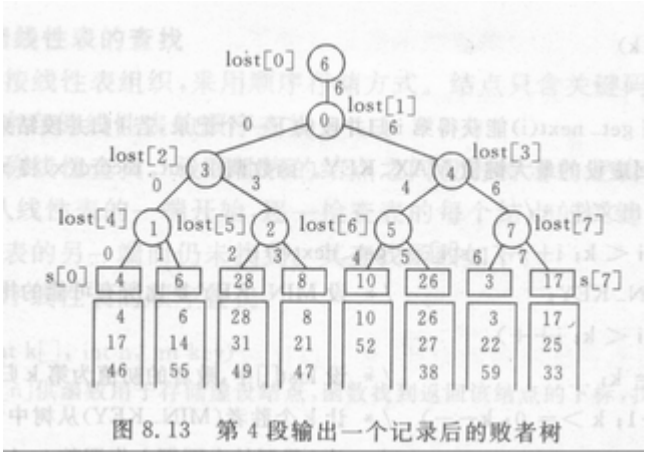


图 8.13 第 4 段输出一个记录后的败者树

【函数】 败者树上的调整。

```
Void adjust (int lost[],int s[],int sp,int k)
{ / * 从叶结点 s[sp]到根结点父结点 lost[0] 调整败者树 * /
    int t,temp;
    t=(sp+k)/2; / * 求 s[sp]的父结点 * /
    while (t>0){
        if (s[sp]>s[lost[t]]){
            temp=sp; / * sp 与 lost[t]交换 * /
            sp=lost[t] ; / *原失败者胜出 * /
            lost[t]=temp;
        }
    t/=2; / * 继续向上* /
    }
    lost[0]=sp; / * sp 为最后优胜者* /
}
```

【函数】 败者树 k 路归并。

```
# define N 32
int lost[N],s[N+1]; / * N 是最大可能的 k 值 * /
/ * lost[0]至 lost[k-1]为败者树的内部结点。s[0]至 s[k]中前 k 项为败者树
的叶结点，k 是虚段号，s[k]将存储虚设的最小值。* /
# define MIN_KEY -1
# define MAX_KEY 9999
int k_merge (int k)
{ int i, t;
/ *设函数调用 get_next(i)能获得第 i 归并段的下一个记录，若 i 归并段结束
时，函数返回虚设的最大键值 MAX_KEY。函数调用 put_record(x)将 x 所指记录
写到输出文件 * /
    for (i=0; i<k : i++) s[i] = get _next(i);
    s[k] = MIN_KEY;/* 设 MIN_KEY 是比所有可能的排序码都小的值 * /
```

```
    for(i=0;i<k;i++)
          lost[i]=k; / * 设 lost[ ]中，败者的初值为第 k 归并段 */
    for(i=k-1;k>=0;k--)/*让 k 个胜者 (MIN_KEY )从树中调整出去*/
        adjust (lost,s,I,k); / *从 i 段开始自下而上作调整  * /
    while (s[lost[0]]!=pass with flying colors_KEY ) {
        t=lost[0];put_record(&s[t]);s[t]=get_next(t);
        adjust(lost,s,t,k);
    }
}
```

为减少外排序的归并趟数，增加初始段的长度是一个值得考虑的方案，常采用以下所述的转换－选择排序方法产生初始段。利用工作数组先从输入文件读入足够多的记录，选出其中最小值记录输出。然后从输入文件读入一个记录置换输出记录。接着反复从数组中选不比前一个输出结点小的最小记录输出。如不存在的记录则输出一个结束标志，以表示当前段已结束，并开始下一个归并段的选择。上述工作直到文件的记录已输入完，且工作数组中的记录也全部输出为止。这样做的好处是一般能得到比工作数组大小更长的初始段。这是因为输入置换的记录有可能会比当前刚输出的记录更大，它也会被输出在同一初始段中。

8.9 查找

查找就是在按某种数据结构形式存储的数据集合中，找出满足指定条件的结点，按查找的条件分类，有按结点的关键码查找、关键码以外的其他数据项查找或其他数据项的组合查找等。按查找数据在内存或外存，分内存查找和外存查找。按查找的目的，查找只是为了确定指定条件的结点存在与否，称为表态查找；查找是为确定结点的插入位置或为了删除找到的结点，称为动态查找.为了叙述简便，这里假定仍按结点的关键码查找，而查找值称为查找码。

1. 顺序存储线性表的查找

设结点集合按线性表组织，采用存储方式。结点只含关键码并且是整数。

（1） 顺序存储线性表的顺序查找

顺序查找也称线性查找。设线性表的结点之间没有特定的逻辑顺序，顺序查找是对给定的查找码，从线性表的一端开始，逐一检查表的每个结点的关键码，直至所找的结点找到或到达线性表的另一端而仍未找到。若函数如下：

【函数】 无序线性表的顺序查找。

```
Int search1 (int k[],int n,int key)
{ / * 设 k [n ] 供函数用于存储虚设结点，函数找到返回该结点的下标，找不到返回-1 * /
    int i;
```

```
    k[n]=key;
    for i=0;key!=k[i];i++);
    returm i<n?i:-1;
}
```

（2） 有序顺序存储线性表的顺序查找

假定线性表按结点键值递增的顺序存放于线性表中，则顺序查找时，发现结点的键值比查找码大时，就可提早得出没有要找结点的结论，相应函数如下：

```
int search2 (int k[],int n,int key)
{ / *设 k[n] 供函数用于存储虚设结点，函数找到返回该结点的下标，找不到返回-1 * /
    int i;
    k[n]=MAX_KEY; / * 设所有结点的键值都小于 MAX_KEY * /
    for (i=0;key>k[i];i++);
    if (i<n&&k[i]==key)return i ;
    return -1;
}
```

（3） 有序顺序存储线性表的二分法查找

假定线性表按结点键值递增的顺序存放于线性表中，则可以改进顺序查找的方法。顺序查找时，每比较一次，查找范围只减少 1，二分法查找能大大减少查找范围。

设分别用 low 和 high 表示当前查找范围的下界和上界，首选计算查找范围的中间位置：mid=(low+high)/2。用查找码与位于 mid 位置的结点比较，发现查找码大于它时，表示新的查找范围应为[mid+1,high]，若查找码小于它，表示新的查找范围应为[low mid-1]。这样一次比较后，要么因查找码与它相等，查找成功，要么是查找范围缩小一半后，在新查找范围内继续查找。直到查找成功，或出现 high <low，表示不再有查找范围，因而查找失败。二分法查找函数定义如下：

```
int bin_search(int k[],int n,int key)
{ / * 函数找到返回该结点的下标，找不到返回－1 * /
    int low=0,high=n-1,mid;
        while (low<=high) { / *二分循环查找 * /
        mid=(high+low)/2;
        if (key==k[mid]) return mid;
        if (key>k[mid])low=mid+1;
        else high=mid-1;
    }
    retrun - 1;
}
```

（4） 有序顺序存储线性的插值查找

插值查找是对两分法查找的一种改进，依然假定线性表按结点键值递增的顺序存放，并且线性表中结点的键值几乎均匀出现，则可按查找范围两端结点的键值计算出键为查找码的结点的大致位置，如它存在的话。其位置可通过线性插值来计算。

假定分别用 key 和 high 表示当前查找范围的下界和上界，查找码为 key 。首先计算查找范围的插值位置：

```
pos=(key-k[low])/(k[high]-k[low])*(high-low)+low;
```

用 key 与 k [pos]比较，发现结点 key > k [pos]，表示新的查找范围应为[pos + 1 , high] ，若 key < k [pos]，表示新的查找范围应为[low , pos -1]。这样一次比较后，要么是查找成功(key = = k [pos]) ，要么是查找范围缩小后，在新的查找范围内继续查找。直到查找成功，或出现 high < low ，表示已没有可找范围，因而查找失败。插值查找函数定义如下：

```
int ins_search(int k[],int n,int key)
{ / * 函数找到返回该结点的下标，找不到返回－1 * /
    int low=0,high=n-1,pos;
    while (low<=high) {
    pos=(key-k[low])/(k[high]-k[low])*(high-low)+low;
        if (key==k[pos]) return pos ;
        if (key>k[pos])low=pos+1;
        else high=pos-1;
    }
    retrun - 1
}
```

2. 分块查找

一般说来，顺序存储的线性表，比较适宜于静态查找，前面算法也都是静态查找的方法。为提高查找速度，将线性表保持有序，并用二分法查找，对提高查找效率是非常有效的。但当数据集上有频繁的插入或删除操作时，为能继续保持有序需要大量移动数据存储位置的操作。将数据分块组织，块与块之间有序，块内数据任意，是一种比较适合用动态查找的数据组织形式。

例如将线性表分成 n 块，块内数据可以任意，但块间数据是有序的，前一块中任一结点的键值都小于后一块中所有结点的键值。另外，还要求建立一个索引表，每个索引五项原则对应表中的一块，至少有这样两项信息，它们是对应中结点的最大键值和块的第一个结点的指针，索引表中的索引项按其键值有序存储。

分块查找的过程分两步，先用二分法在索引表中查索引项，确定要查的结点在哪一块。然后，再在相应块内顺序查找。由于块内结点无序，多数情况增删比较简便，不需要大量的移动结点。删除可做标记，增添在块内任意位置。

3. 链接存储线性表的查找

链接存储线性表上的查找只能从链表的首结点开始顺序查找。设链表结点除整数关键码外，另只含一个后继结点指针：

```
typedef struct node { / * 链表的表元类型 * /
    int key;
    struct node * next ;
} NODE;
```

（1） 无序链表的查找

无序链表的结点之间没有特定的逻辑顺序，对给定的查找码，从链表的首结点开始，逐一检查表的每个结点的关键码，直至所找的结点找到或直到考察了链表的末结点后仍未找到。其查找函数定义如下：

【函数】 无序链表的查找。

```
NODE * search1(NODE *h,int key)
{ / * 设 h 是链表的首结点指针，找到时，函数返回该结点的指针，否则返回
NULL * /
    for (;h!=NULL&&h->key;h=h->next);
    return h;
}
```

（2）有序链表的查找

假定链表按结点键值递增的顺序链接，则顺序查找时，发现结点的键值比查找码大时，就可提早得出没有要找结点的结论，相应函数定义如下：

【函数】 有序链表的顺序查找。

```
NODE * search2(NODE * h,int key)
{ / * 设 h 是链表的首结点指针，找到时，函数返回该结点的指针，否则返回
NULL * /
    for (;h!==NULL&&h->key<key;h=h->next);
    return (h==NULL||h->key!=key)?NULL:h;
}
```

（3） 序链表的动态查找

假定链表按结点键值递增的顺序链接，且查找是为了增删结点。如为了删，则在找到删除结点后，为了能删除该结点，需要知道该结点的前驱结点指针。同样地，如查找是为了插入结点，为能正确完成插入，需要知道插在哪一个结点之后，为此，动态查找需返回两个指针：确定查找结束的结点指针和该结点的前驱结点指针。另外，函数还要区分在首结点时找到和直至到了未结点也未找到的情况。为此，函数增设能带回上述信息的两个指针参数，分别为 p 和 q，其中*q 为确定查找结束的结点指针，＊p 为它的前驱结点指针。查找结果可按两指针结果值的不同情况来区分：

当* p==NULL 且* q!==NULL，在的首结点找到，＊q 指向链表的首结点；

当* p!==NULL 且* q!==NULL，在链表的某个中间位置找到，*q 指向结束查找时的结点，＊p 指向它的前驱结点。

当* p!==NULL 且* q==NULL，直找至链表的末尾，＊p 指向链表的末结点；

当* p==NULL 且* q==NULL，所找链表是空链表。

【函数】 有序链表的动态查找。

```
Int search (NODE * h,int key, NODE * * p, NODE * * q)
{ / * 设 h 是链表的首结点指针，找到时，函数返回 1，否则返回 0  * /
NODE * u=NULL,* v=h;
For (;v!=NULL&&v->key<key;u=v,v=v->next);
p=u;* q=v;
return v!=NULL&&v->key==key;
}
```

以下是说明上述函数用法的示意程序。

【例 8．18】一个应用动态查找的示意程序。程序输入整数建立有序链表，然后逐一输入要删除结点的键值，将相应结点从链表中删除，直至链表为空结束。

【例 8．18 程序】

```
#include<stdio.h>
#include<stdiob.h>
typedef struct node { / * 链表表元类型 * /
    int key;
    struct node * next;
}NODE;
int search(NODE * h,int key,NODE ** p,NODE ** q)
{ / * 设 h 是链表的首结点指针 * /
    NODE * u=NULL,* v=h;
    For(;v!=NULL&& v->key<key;u=v,v=v->next);
```

```
    p=u;*q=v;
    return v!=NULL && vgkey==key;
}
void write(NODE * h)
{    while(h!=NULL){ / *顺序考察链表表元，至链表为空 * /
    printf("%4d",h->key);
    h=h->next;
}
    printf("\n");
}
void main()
{    int t,n,I;
    NODE * x,* y h=NULL ,* P;
/ *输入 n 个整数，建立一个有序链表 * /
    printf("Enter n.\n");scanf("%d",&n);
    for(I=0;I<n;I++){
        scanf("%d",&t);
        p=(NODE *)malloc(sizeof(NODE));
        p->key=t;p->next=NULL;
        search(h,t,&x,&y); / * 寻找插入点 * /
        if(x==NULL){ / * 插在首结点之前 * /
            p->next=h;h=p;
        }
        else{ / 插在*x 结点之后 * /
            p->next=x->next;x->next=p;
        }
    }
    write(h);
    do{ / *逐一删去链表中的结点，直至链表为空 * /
        printf("Enter delete key .\n");
        scanf("%d",&t);
        if(search(h,t,&x,&y)){ / *链表中找到值为 t 的结点 * /
            if(x==NULL)h=y->next;
            else x->next=y->next;
            free(y);
            write(h); / *让你验证 * /
        }
        else printf("Not found !\n");
     }while (h);
     printf("The linked list if empty!\n");
}
```

4．散列表的查找

散列表又称杂凑表，是一种非常实用的查找技术。由于查找码与结点在数据结构中的位置之间不存在确定关系，查找只能反复通过查找码与结点的关键码的比较来实现，能过比较缩小的查找范围。提高查找效率的途径是通过一次比较能大幅度减小查找范围。理想的情况是直接利用查找码的信息，一次或几次存取便能存取的查结点。为此必须在结点的存储位置和它的关键码间建立一个确定的关系，从而让查找码直接利用这个关系确定结点的位置。

在散列表中，结点存储时，以结点的关键码为自变量，按某种公式计算其存储位置之不理，并以该位置存储。查找时，以查找码为自变是量计算同一公式，就能得到应结点的存储位置，去取结点，所以散列表能进行快速查找。称由查找码计算结点存储位置的公式为散列函数，记为 H。

一般情况下，能存入散列表中结点的所有可能的不同关键码个数比散列表能存储的结点个数要大得多，因此可能会出现两个不同的关键码 k1 和 k2 它们的散散列函数值相等：H(k1)=H(k2)且 k1!=k2，这种现象称为冲突，并称 k1 和 k2 为同义词。为了有效地使用散列技术，需要解决两个问题：

• 找一个好的散列函数，使出现冲突的机会尽可能少；

• 设计有效的解决冲突的方法。

设散列表最多能存储→个结点，每个散列函数 H 的值对应一个存储区间，称为桶，每个桶可以存放 w 个结点，为散列表开设 m 个桶，称为基本区，散列函数的值域为[0，m-1],通常 m 比 n/w 要大约 20%。

（1） 常见的散列函数

① 质数除取余法：令 p 是接近 M 的质数，设查找码为 KEY，要求的桶号为 T，计算 T 的散列函数为
t=key % p

② 基数转换法：把查找码看作是某个基数制上的整数，然后将它转换成另一基数制上的数。如把一个十进制数 321 看作是十三进制的，即：
321（13）=（3*13+2）*13+1=534

③ 平方取中法：让查找码自乘，取其中间若干位的值，并乘上某个比例因子，使值在 0 至 m-1 之间。

④ 折叠法：当查找码较长进，可将查找码分成多段，左边的段向右折，右边段向左折，然后将它们叠加。

⑤ 移位法；当查找码较长时，可将查找码分成多段，左边的段向右移，右边段向左移，然后将它们叠加。

实际应用时，常将上述多种方法综合使用。

（2） 解决冲突的方法

解决冲突方法是设法在散列表中找一个空位。一般来说，影响冲突在三个因互：

• 散列函数是否均匀分布；

• 处理冲突的方法；

• 负载因子的大小。

负载因子是指散列表中已被结点所占用空间与散列表整个空间的比。设散列表当前已存有 n 个结点，则 n/→为这个表的负载因子 a。负载因子的大小对发生冲突有重要影响。

这里介绍几种常用的解决冲突的方法。

① 线性探查法：如按查找码 key 求得第 H（key）桶已满时，就得到另一个桶去找空位，将结点存入。线性探查法使用下列探查序列：
H(key),H(key)+1,H(key)+2,…, m-1,0,1, …,H(key)-1

使用线性探查法解决冲突，会出现堆积现象。可能会发生这样的情况，由某结占的关键码计算出它的散列地址，希望按该地址存储该结点，但该位置已被不是它的同义词的结点所占用。因冲突沿着线性探查序列进行查找，而这些位置也可能已存入非同义词的其他结点，结果造成不是同义词的许多结点处在同一个探查序列中。

② 双散列函数法：由于线性探查法容易产生堆积现象，会大在降低查找效率，可用双散列函数法。由散列函数 H1（）产生 0 至 m-1 的桶号 d，当冲突发生进，散列函数 H2（）产生 1 列 m-1 之间并与 m 互质的数 c 作为对 H1（）的补偿，使探查序列跳跃式地分布在整个散列表中：
（d+c）%m、(d+2c)%m、(d+3c)%m、…

线性探查法和双散列函数法总称开始地址法，用开发地址法处理冲突的要步骤为：

• 按散列函数计算桶号 I。

• 到第 I 号桶查找，若查到则查找成楞，查找过过程结束。若未查到且桶未满，则查找失败结束。

• 若当前查不到，且桶是满的，则按选定的探查序列探查，或者到成功结束或查不到结束

③ 拉链法：用拦链法处理冲突，要求散列表的每个结点增加一个指针字段，用于链接同应义词的子表，链表中的结点都是同义词。查找方法哪下：

• 按散列函数计算出桶号 I。

• 判断对应的同义词链表指针是否为空，若为空则查找失败结束。

• 遍历该同义词链表，查到则成功结束，若直到表尾仍未找到，则查找失败。

【例 8．19】在编译程序中，需要频繁地查语言的关键字，为提高查找速度，通常给语言的全部关键字建立一个散列表。本例以 C 语言为例，说明散列表的用法。

设各关键字的查找是等概率的，采用双散列函数法解决冲突，取负载因子 a<0.7. 因 c 语言的关键字有 32 个，另仅为了说明散列表应用的原理，令每个桶只能存一个结点即 w=1，所以取散列表大小 N=47。

另对双散列表函数作如下考虑；H1 采用将查找码每三个字符为一段折叠后累计，并用质数除取余法，所有制数为 47。H2 采用将查找码每两个字符为一段折叠后累计并用质数除取余后加 1，所有质数为 43。

下面的程序输出建立的关键字表和建表过程各关键字发生冲突的情况。

【例 8．19 程序】

```
#include<stdio.h>
#include<string.h>
#define → 47
#define P1 47
#define p2 43
#define LEN 10
char tbl[N][LEN];
int H1(char * key)
{    long k=0,d;
      int c;
      while(*key){ / *每三个字符为一段折叠后累计 * /
            for(d=c=0;* key!='\0' && c<3;c++)
                  d=(d<<8)+*key++;
            k+=d;
      }
      return k% p1; / * 质数除取余法 * /
}
int H2(char * key)
{    long k=0,d;
      int c;
      while (*key){ / *每两个字符为一段折叠扣累计 *  /
            for(d=c=0;* key!='\0'&& c<2;c++)
                  d=(d<</8)+*key ++;
            k+=d;
```

```
      }
      return k%p2+1; / * 质数除取余后加 1 * /
}
char* kWrod[]={"auto","break","case","char","const",
            "continue","default","do","double","else",
            "enum","extern","float","for","goto",
            "if","int","long","register","return",
            "short","signed","sizeof","static","struct",
            "switch","typedef","union","unsigned","void",
            "volatie","while"};
void main()
{   int count[N],I,pos,c;
      for(I=0;I<→;I++){ / * 散列表清空 * /
            count[i]=0; / * 各位置的冲突计算器清 0 * /
            tbl[i][0]='\0' / * 各存储位置空 * /
      }
      for(I=0;I<sizeof k word/sizeof k word[0];I++){ / *关键字存入散列
表 * /
            pos=H1(kword[i]);
            c=H2(kWord[i]);
            while(tbl[pos][0]!='\0\&&strcmp(tbl[pos],kword[i])){
                  count[pos]++; / *冲突计数增 1 * /
                  pos =(pos+c)% →; / *双散列解决冲突 * /
            }
            strcpy(tbl[pos],kWord[i]);
      }
      for(I=c=0;I<→;I++)
            if(tbl[i][0]!='\0')
                  printf("10s%%c",tbl[i],++c%5==0?'\n':'');
      printf("\n\n");
      for(I=c=0;I<→;I++)
            if (count[i])
                  printf("%4d%c",cont[i],++c%5==0?'\n':'\t')';
      printf("\n\n");
}
```

第 9 章 常用算法设计方法

要使计算机能完成人们预定的工作，首先必须为如何完成预定的工作设计一个算法，然后再根据算法编写程序。计算机程序要对问题的每个对象和处理规则给出正确详尽的描述，其中程序的数据结构和变量用来描述问题的对象，程序结构、函数和语句用来描述问题的算法。算法和数据结构是程序的两个重要方面。

算法是问题求解过程的精确描述，一个算法由有限条可完成机械地执行的、有结果指令组成。指令正确地描述了要完成的任务和它们所执行的顺序。计算机按算法指令所描述的顺序执行算法的指令能在有限的步骤内终止，或终止于给出问题的解，或终止于指出问题对此输入数据无解。

通常求解一个问题可能会有多种算法可供选择，选择的主要标准首先是算法的正确性和可靠性，简单性和易理解性。其次是算法所需要的存储空间少和执行更快等。

算法设计是一件非常困难的工作，经常采用的算法设计技术主要有迭代法、穷举搜索法、递推法、贪婪法、回溯法、分治法和动态规划法等等。另外，为了以更简洁的形式设计和描述算法，在设计算法时又常常采用递归技术，用递归描述算法。

9.1 迭代法

迭代法是用于求方程或方程组近似根一种常用算法设计方法。设议程为 f(x)=0，有某种数学方法导出等价的形式 x=g(x)，然后按以下步骤执行：

（1） 选一个方程的近似根，赋给变量 x_0；

（2） 将 x_0 的值保存于变量 x_1 然后计算 $g(x_1)$，并将结果存于变量 x_0；

（3） 当 x_0 与 x_1 差的绝对值还小于指定的精度要求时，重复步骤（2）的计算。

若方程有根，并且用上述方法计算出来的近似根序列收敛，则按上述方法求得的 x_0 就认为是方程的根。上述算法用 C 程序形式书写有：

【算法】迭代法求方程的根

```
{x0=初始近似根;
  do{
    x1=x0;
    x0=g(x1);                          / * 按特定的方程计算新的近似根 * /
    }while(fabs(x0-x1)>Epsilon);
    printf("方程的近似根是%f\n", x0);

}
```

迭代算法也常用于求方程组的根，令

$X=(x_0, x_1, \cdots, x_{n-1})$

设方程组为：

$x_i=g_i(X)$ (I=0，1，…，n-1)

则求方程组根的迭代算法可描述如下：

【算法】迭代法求方程组的根。

```
{for(i=0; i<n; i++)
    x[i]=初始近似根;

do{
  for(i=0; i<n; i++)
      y[i]=x[i];                        / * 保护老的近似根 * /
  for(i=0; i<n; i++)
      x[i]=gi(x);                       / * 按特定的方程组计算新的近似根 * /
  for(delta=0.0, i=0; i<n; i++)        / * 求新老近似根的最大误差 * /
    if(fabs(y[i]-x[i])>delta)delta}=fabs(y[i]-x[i]);

}while(delat>Epsilon);
for(i=0; i<n; i++)
  printf("变量 x[%d]的近似根是%f", i, x[i]);
printf("\n")';

}
```

具体使用迭代法求根时应注意以下两种可能发生的情况：

（1） 如方程无解，算法求出的近似根序列就会不收敛，迭代过程会变成"死循环"，因此在使用迭代算法前应先考察方程是否有解，并在程序中对迭代的次数给予限制。

（2） 方程虽有解，但迭代公式选择不当，或迭代的初始近似根选择不合理，也会导致迭代失败。

9.2 穷举搜索法

穷举搜索法是对可能是解的众多候选解按某种顺序进行逐一枚举和检验，并从中找出那些符合要求的候选解作为问题的解。

【例 9. 1】将 A，B，C，D，E，F，G，H 这 8 个变量排成如图 9.1 所示的一个大梯形和四个小梯形，这 8 个变量分别取[1，8]上的整数，且均不相同。求使如图所未一个大梯形和四个小梯形四个角上的变量和全相等的全部解。如图 9.1（b）就是一个解。

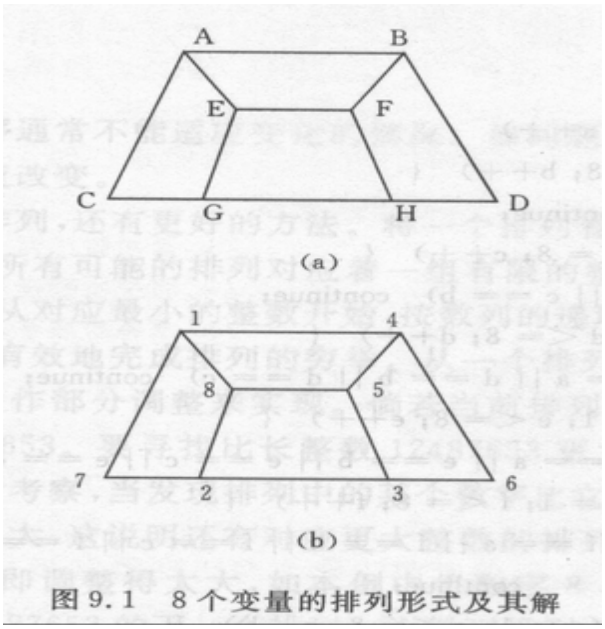


图 9.1 8 个变量的排列形式及其解

程序引入变量 a.b.c.d.e.f.g.h 并让它们分别顺序取 1 至 8 的整数。在它们互不相同的条件下，测试由它们排成的如图 9.1（a）所示的五个梯形四个角上的变量之和是否相等，如相等，即为一种要求的排列，把它们输出。当这些变量取尽所有的组合后，程序就得到全部可能的解，细节见下面的程序。

【例 9. 1 程序 1】

```
#include<stdio.h>
#define SIDE_N     5
#define LENGTH     4
#define VARIABLES  8
int a, b, c, d, e, f, g, h;
int * pt[]={&a, &b, &c, &d, &e, &f, &g, &h};
int *side[SIDE_N][LENGTH]={&a, &b, &c, &d,
                           &e, &f, &g, &h,
                           &a, &b, &e, &f,
                           &a, &c, &e, &g,
                           &b, &d, &f, &h};
char * format[VARIABLES]={"%8d", "%11d\n\n", "%12d", "%4d\n\n\n",
                          "%4d", "%7d", "%7d",  "%7dn\n\n"};
int * va[VARIABLES]={&a, &b, &e.&f, &c, &g, &h, &d};        / * 输出顺序 * /
int side_total [SIDE_N];
void mina()
{  int i, j, t, equal;
   for(a=1; a<=8; a++)                                     / * 所有可能的 a * /
     for(b=1; b<=8; b++){                                  / * 所有可能的 b* /
        if (b==a)continue;
        for(c=1; c<=8; c++) {                              / * 所有可能的 c * /
           if(c==a||c==b)continue;
           for(d=1; d<=8; d++){                            / * 所有可能的 d * /
```

```
            if(d==a||d==b||d==c)continue;
            for(e=1; e<=8; e++) {                          / * 所有可能的 e * /
               if(e==a||e==b||e==c||e==d)continue;
               for(f=1; f<=8; f++) {                       / * 所有可能的 f * /
                  if(f==a||f==b||f==c||f==d||f==e)continue;
                  for(g=1; g<=8; g++){                     / * 所有可能的 g * /
                     if(g==a||g==b||g==c||g==d||g==e||g==f)continue;
                  h=36-(a+b+c+d+e+f+g); /*a+b+c+d+e+f+g+h=36 * /
                  for(i=0; i<SIDE_N; i++){                 / *求各组数字之和 * /
                      for(t=j=0; j<LENGTH; j++)
                          t+=* side[i][j];
                      side_total[i]=t;
                  }
                  for(equal=1, i=0; equal && i<SIDE_N-1; i++)
                       / *确定各组之和是否相等 * /
                    if(side_total[i]!=side_total[i+1]equal=0;
                    if(equal){                     / *各组都相等，输出找到的解 * /
                        for(i=0; i<VARIABLES ; i++);
                            printf(format[i], *va[i]);
                        scanf("% * c");
                    }
                  }
               }
            }
         }
      }
   }
}
```

按穷举法编写的程序通常不能适应变化的情况。如问题改成有更多变量排成梯形，程序的循环重数就要相应改变。

对一组数穷尽所有排列，还有更好的方法。将一个排列看作一个长整数，则每个排列对应一个不同的长整数，所有可能的排列对应着一组有限的整数。将这组整数按从小到大的顺序排成一个数列，从对应最小的整数开始，按数列的递增顺序逐一列举每个排列对应的每一个整数，这能更有效地完成排列的穷举。从一个排列找到对应数列下的一个排列可在当前排列的基础上作部分调整实现。倘若当前排列为 1、2、4、8、7、6、5、3，并令其对应的长整数为 12487653。要寻找比长整数 12847653 更大的排列，可从该排列的最后一个数字顺序向前逐位考察。当发现排列中的某个数字比它前一个数字大时，如本例中的 8 比它的前一个数字 4 大，这说明还有对应更大整数的排列。但为了顺序从小到大列举出所有的排列；不能立即调整得太大，如本例中将数字 8 与数字 4 交换得到的排列 12847653 就不是排列 12487653 的下一个排列。为得到排列 12487653 的下一个排列，应从已考察过的那部分数字中选出比数字 4 大，但又是它们中最小的那个数字，比如数字 5。该数字也是从后向前考察过程中第一个比 4 大的数字。5 与 4 交换后，得到排列 12587643。在前面数字 1、2、5 固定情况下，还应选择对应最小整数的那个排列。为此还需要将后面那部分数字的排列顺序颠倒，如将数字 8、7、6、4、3 的排列顺序颠倒，得到排列 1、2、5、3、4、6、7、8 的下一排列，按以上想法编写程序如下。

【例 9．1 程序 2】

```
#include<stdio.h>
#define SIDE_N      5
#define LENGTH      4
#define VARIABLES   8
int A, B, C, D, E, F, G, H;
int * pt[]={&A, &B, &C, &D, &E, &F, &G, &H};
int *side[SIDE_N][LENGTH]={&A, &B, &C, &D,
                           &E, &F, &G, &H,
                           &A, &B, &E, &F,
                           &A, &C, &E, &G,
                           &B, &D, &F, &h};

char * format[VARIABLES]={"%8d", "%11d\n\n", "%12d", "%4d\n\n\n",
                          "%4d", "%7d", "%7d", "%7d", "%7dn\n\n"};


int * va[VARIABLES]={&A, &B, &E.&F, &C, &G, &H, &D};
int side_total [SIDE_N];
void main()
{  int i, j, t, equal;
   for(j=0; j<VARIABLES; j++)
       *pt[j]=j+1;                    / *置初始排列 1, 2, 3, 4, 5, 6, 7, 8 * /
   while(1){                          / *穷举所有排列循环 * /
      for(i=0; i<SIDE_N; i++) { / * 求各组数字之和 * /
         for(t=j=0; j<LENGTH;  j++)
             t+=*side[i][j];
         side_total[i]=t;
      }
      for(equal=1, i=0; equal && i<SIDE-N-1; i++)
       / *确定各组之和是否相等 * /
        if (side_total[i]!=side_total[i+1])equal=0;
      if(equal){                        / * 各组都相等, 输出找出的解 * /
        for(i=0; i<VARIABLES; i++)
           printf(format[i], *va[i]);
        scanf("% * c");
      }
      / * 寻找下一个排列交换点 * /
      for(j=VARIABLES -1; j>0; j--)
        if(* pt[j]>*pt[j-1])break;
      if(j==0)break;                   / *已没有更大的排列 * /
      for(i=VARIABLES-1; i>=j; i--)    / *寻找交换元素 * /
         if(*pt[i]>pt[j-1])break;
      t=*pt[j-1];  * pt[j-1]= *pt[i];  * pt[i]=t;   / *交换 * /
```

```
        for(i=VARIABLES-1; i>j; i--, j++){          / *后部分元素颠倒 * /
            t= * pt[j]; *pt[j]=*pt[i]; pt[i]=t;
        }
    }
}
```

9.3 递推法

递推法是利用问题本身所具有的一种递推关系求问题解的一种方法。设要求问题规模为 N 的解，当 N=1 时，解成为已知，或能非常方便地得到解。能采用递推法构造算法的问题有重要的递推性质，即当得到问题规模 i-1 的解后，由问题的递推性质，能从已求得的规模为 1，2，…， i-1 的一系列解，构造出问题规模为 I 的解。这样，程序可从 i=0 或 i=1 出发，重复地，由已知至 i-1 规模的解，通过递推，获得规模为 i 的解，直至得到规模为 N 的解。

【例 9.2】编写程序，对给定的 n(n≤100)，计算输出 k 的阶乘 k！（k=1，2，…，n）的全部有效数字。

由于要求的整数可能大大超出一般整数的位数，程序用一维数组存储长整数，存储长整数数组的每个元素只存储长整数的一位数字。如有 m 位长整数 N 用数组 a[]存储：

$N=a[m]\times10^{m-1}+a[m-1]\times10^{m-2}+\cdots+a[2]\times10^{1}+a[1]\times10^{0}$

并用 a[0]存储长整数 N 的位数 m，即 a[0]=m。按上述约定，数组的每个元素存储 k 的阶乘 k!的一位数字，并从低位到高位依次存于数组的第二个元素、第三个元素……例如 5！=120，在数组中的存储形式为：

3	0	2	1	……

首元素 3 表示长整数是一个 3 位数，接着是低位到高位依次是 0、2、1，表示长整数为 120。

计算阶乘 k！可采用以已求得的阶乘（k-1）！连续累加 k-1 次后求出。例如，已知 4！=24 计算 5！，可对原来的 24 再累加 4 次 24 后得到 120。细节见以下程序。

【例 9. 2 程序】

```
#include<stdio.h>
#include<malloc.h>
#define MAXN 1000
void pnext(int a[] , int k)                  / *已知 a 中的(k-1)！求出 k!于 a */
{  int *b , m=a[0], i, j, r, carry;
   b=(int * )malloc(sizeof(int)* (m+1));
   for(i=1; i<=m; i++)b[i]=a[i];
   for(j=1; j<k; j++){                      / *控制累加 k-1 次 * /
       for(carry=0, i=1, i<=m; i++){
           r=(i<=a[0]?a[i]+b[i]: a[i])+carry;
```

```
            a[i]=r%10;  carry=r/ 10;
        }
        if(carry)a[++m]=carry;
    }
    free(b);
    a[0]=m;
}
void write(int * a, int k)
{   int i;
    printf("% 4d!=", k);
    for(i=a[0]; i>0; i--)printf("%d", a[i]);
    printf("\n\n");
}
void main()
{ int a[MAXN], n, k;
  printf("Enter the number n: "); scanf ("%d", &n);
  a[0]=1; a[1]=1; write(a, 1);
  for(k=2; k<=n; k++){
      pnext(a, k);
      write(a, k);
      getchar();
  }
}
```

9.4 递归

递归是设计和描述算法的一种有力的工具，由于它在复杂算法的描述中经常被采用，为此在进一步介绍其他算法设计方法之前先讨论它。

能采用递归描述的算法通常有这样的特征：为求解规模为 N 的问题，设法将它分解成一些规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模稍大问题的解。特别地，当规模 N=1 时，能直接得到解。

【例 9．3】编写计算斐波那契(Fibonacci)数列的第 n 项函数不清 fib(n)。
斐波那契数列为：0，1，1，2，3，…即：

fib(0)=0
fib(1)=1
fib(n)=fib(n-2)+fib(n-1) (当 n>1 时)

写成递归函数有：

【例 9．3 函数】

```
int fib(int n)
{    if (n == 0) return 0;
     if (n == 1) returen 1;
     if(n > 1) return fib(n-2) + fib (n -1);
}
```

递归算法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题（规模为 n）的求解推到比原问题简单一些的问题（规模小于 n）的求解。例如上例中，求解 fib(n)，把它推到求解 fib(n-2)和 fib(n-1)。也就是说，为计算 fib(n)，推到计算 fib(n-2)和 fib(n-1)，而计算 fib(n-2)和 fib(n-1)，又把它们推到计算 fib(n-4)和 fib(n-3)。依次类推，直到计算 fib(0)和 fib(1)，分别能立即得到结果 0 和 1。在递推阶段，必须要有终止递归的情况，例如在函数 fib 中，当 n 为 0 和 1 的情况。

在回归阶段，当获得最简单情况的解后，逐级返回，依次获得稍复杂问题的解。例如得到 fib(0)为 0 和 fib（1）为 1 后，返回获得 fib(2)的结果……得到了 fib(n-2)和 fib(n-1)的结果后，返回获得 fib(n)的结果。

在编写递归函数时要注意，函数中的局部变量和参数只是局限于当前调用层的，当递推进入"简单问题"层时，原来层次上的参数和局部变量便隐藏起来。在一系列"简单问题"层，它们各自有自己的参数和局部变量。

由于递归引起一系列的函数调用，并且可能会有一系列的重复计算，递归算法的执行效率相对较低。当某个递归算法能较方便地转换成递推算法时，通常按递推算法编写程序。例如上例斐波那契数列的第 n 项的函数 fib(n)应采用递推算法，即从斐波那契数列的前两项值出发，逐次由前两项计算出下一项，直至计算出要求的第 n 项。

【例 9．4】找出从自然数 1，2，…，n 中任取 r 个数的所有组合。例如 n=5，r=3，所有组合为：

```
5 4 3
5 4 2
5 4 1
5 3 2
5 3 1
5 2 1
4 3 2
4 3 1
4 2 1
3 2 1
```

分析例示所列的 10 个组合，可用这样的递归思想来考虑求组合函数的算法，设函数为 void comb(int m, int k)，即找出从自然数 1，2，…m 中任取 k 个数的所有组合。当组合的第一个数字选定时，其后的数字是从余下的 m-1 个数中取 k-1 个数的组合。这就将求 m 个数中取 k 个数的组合问题，转化成求 m-1 个数中取 k-1 个数的组合问题。设函数引入工作数组 a[]存放求出的组合的数字，约定函数将确定的 k 个数字组合的第一个数字放在 a[k]中，当一个组合求出后，才将 a[]中的一个组合输出。第一个数可以是 m，m-1，…k，函数将确定组合的第一个数字放入数组后，有两种可能的选择，因还未确定组合的其余元素，继续递归去确定组合的其余元素；或因已确定了组合的全部元素，输出这个组合。细节见以下程序中的函数 comb。

【例 9．4 程序】

```
#define <stdio.h>
#define MAXN 100
int a[MAXN];
void comb(int m,  int k)
{   int i, j;
    for(i = m;  i>= k;  i--){           /*顺序选组合的最大数*/
       a[k] = i;
       if (k > 1)                      /*组合中还有其他元素*/
         comb(i-1,  k-1);
       else{                           /*找到了一个组合，将组合的元素输出*/
              for(j = a[0];  j >0;  j--) printf("%4d", a[j]);
              printf("\n");
       }
     }
}
void main()
{   a[0] = 3; comb(5, 3);
}
```

【例 9．5】背包问题。有不同价值、不同重量的物品 n 件，求从这 n 件物品中选取一部分物品的选择方案，使选中物品的总重量不超过指定的限制重量，但选中物品的价值之和为最大。

设 n 件物品的重量分别为 w_0，w_1，…，w_{n-1}，物品的价值分别为 v_0，v_1，…v_{n-1}。采用递归寻找物品的选择方案。设前面已有了多种选择的方案，并保留了其中总价值最大的方案数组 option[]，该方案的总价值存在于变量 maxv。当前正在考察新方案，其物品选择情况保存于数组 cop[]。假定当前方案已考虑了前 i-1 件物品，现在要考虑第 i 号物品；当前方案已包含的物品的重量之和为 tw，至此，若其余物品都选择是可能的话，本方案能达到的总价值的期望值设为 tv。算法引入 tv 是当一旦当前方案的总价值的期望值也小于前面方案的总价值 maxv 时，继续考察当前方案变成无意义的工作，应终止当前方案，立即去考察下一个方案。因为当方案的总价值不比 maxv 大时，该方案不会被再考察，这同时保证函数后找到的方案一定会比前面的方案更好。

对于第 i 件物品的选择考虑有两种可能：

（1）考虑物品 i 被选择，这种可能性仅当包含它不会超过方案总重量限制时才是可行的。选中后，继续递归去考虑其余物品的选择。

（2）考虑物品 i 不被选择，这种可能性仅当不包含物品 i 也有可能会找到价值更大的方案的情况。

按以上思想写出递归算法如下：

【算法】 背包问题，找最佳方案。

```
try(物品 i，当前选择已达到的重量和 tw，本方案可能达到的总价值 tv)
{/*考虑物品 i 包含在当前方案中的可能性*/
  if（包含物品 i 是可接受的）{
      将物品 i 包含在当前方案中；
      if(i< n-1)
          try(i+1，tw+物品 i 的重量，tv)；
      else
            /*又一个完整方案，因它比前面的方案好，以它作为最佳方案*/
            以当前方案作为临时最佳方案保存；
      恢复物品 i 不包含状态；
  }
  /*考虑物品 i 不包含在当前方案中的可能性*/
  if(不包含物品 i 仅是可考虑的)
     if(i < n-1)
          try(i+1，  tw，tv - 物品 i 的价值)；
     else
     /*又一个完整方案，因它比前面的方案好，以它作为最佳方案*/
       以当前方案作为临时最佳方案保存；
}
```

为了理解上述算法，特举以下实例。设有 4 件物品，它们的重量和价值见表 9.1。

表 9.1 背包问题实例

	0	1	2	3
重量	5	3	2	1
价值	4	4	3	1

并设限制重量为 7。则按以上算法，图 9.2 表示找解过程。由图 9.2 知，一旦找到一个解，算法就进一步找更好的解。如能判定某个找查分支不会找到更好的解，算法不会在该分支继续查找，而是立即终止该分支，并去考察下一个分支。

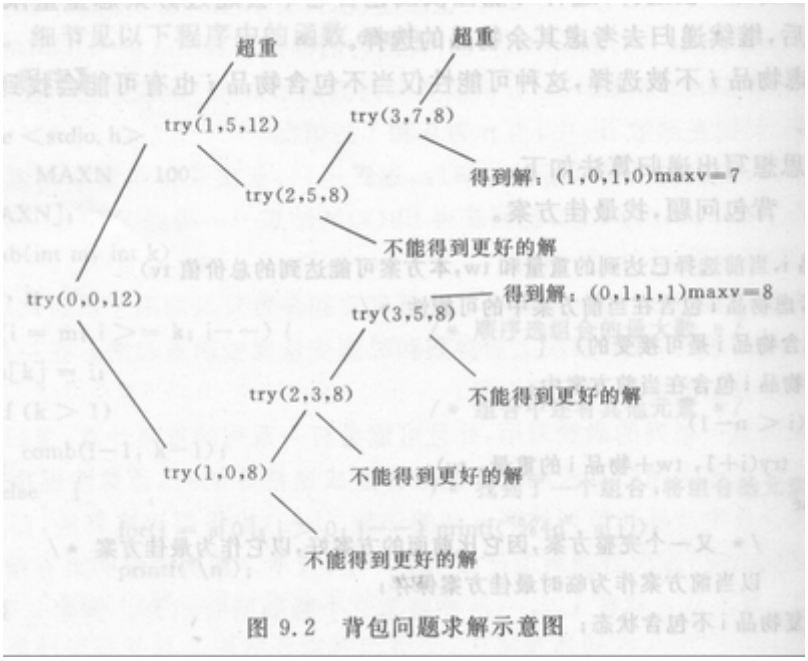


图 9.2 背包问题求解示意图

按上述算法编写函数和程序如下：

【例 9. 5 程序】

```
#include <stdio.h>
#define N 100
double limitW,                          /*限制的总重量*/
          totV ,                        /*全部物品的总价值*/
           maxv;
int option[N],                          /*解的选择情况*/
    cop[N];                             /*当前解的选择情况*/
struct{                                 /*物品的结构信息*/
    double weight;
    double vale;
}a[N];
int n;                                  /*物品种数*/
void find(int i,  double tw,  double tv)
{ int k;
   /*考虑物品 i 包含在当前方案中的可能性*/
   if(tw+a[i].weight <= limitW){      /*包含物品 i 是可接受的*/
      cop[i] = 1;                       /*将物品 i 包含在当前方案中*/
      if( i < n-1) find(i+1, tw+a[i].weight, tv);
      else{       /*又一个完整方案，因它比前面的方案好，以它作为最佳方案*/
          for(k=0;  k< n;  k++)         /*当前方案为临时最佳方案保存*/
               option[k] = cop[k];
          maxv = tv;
      }
```

```
      cop[i] = 0;                    /*恢复物品不包含状态*/
   }
   /*考虑物品 i 不包含在当前方案中的可能性*/
   if(tv-a[i].value > maxv)          /*不包含物品 i 仅是可考虑的*/
      if(i< n-1) find(i+1, tw, tv-a[i].value);
      else{         /*又一个完整方案，因它比前面的方案好，以它作为最佳方案*/
          for(k=0;  k< n;  k++)      /*当前方案作临时最佳方案保存*/
              option[k] = cop[k];
          maxv = tv - a[i].value;
      }
}
void main()
{ int k;
  double w, v;
  printf("输入物品种数\n");
  scanf("%d", &n);
  printf("输入各物品的重量和价值\n");
  for(totV=0.0, k=0; k<n; K++){
     scanf("%lf %lf", &w, &v);
     a[k].weight = w; a[k].value = v;  totV+= v;
  }
  printf("输入限制重量: \n"); scanf("%lf", &limitW);
  maxv =0.0;
  for(k=0; k<n; k++) cop[k]=0;
  find(0, 0.0, totV);
  for(k=0; k<n; k++)
     if(option[k]) printf("%4d", k+1);
  printf("\n 总价值为%.2f\n", maxv);
}
```

作为对比，下面以同样的解题思想，考虑非递归的程序解。为了提高找解速度，程序不是简单地逐一生成所有侯选解，而是从每个物品对侯选解的影响来形成值得进一步考虑的侯选解。一个侯选解是通过依次考察每个物品形成的。对物品 i 的考察有这样几种情况：当该物品被包括在侯选解中依旧满足解的总重量的限制，该物品被包括在侯选解中是应该继续考虑的；反之，该物品不应该包括在当前正在形成的侯选解中。同样地，仅当物品不被包括在侯选解中，还是有可能找到比目前临时最佳解更好的侯选解时，才去考虑该物品不被包括在侯选解中；反之，该物品不包括在当前侯选解中的方案也不应继续考虑。对于任一值得继续考虑的方案，程序就去进一步考虑下一个物品。

【例 9. 5 程序 2】

```
#include <stdio.h>
#define N 100
double limitW;
int cop[N];          /*临时最佳解的物品选择方案，当 opts[i]为 1，物品 i 在解中*/
struct ele{
```

```
        double weight;
        double value;
}a[N];                /*物品的重量和价值信息*/
int k, n;
struct{int flg;       /*物品的考虑转台；0：不选，1：将被考虑，2：曾被选中*/
        double tw;    /*已达到的总重量*/
        double tv;    /*期望的总价值*/
}twv[N];           /*当前侯选解中各物品的考虑和选择状态，以及至该物品侯选解的状态*/
void next(int i, double tw, double tv) /*将考虑 i 号物品*/
{ twv[i].flg =1; twv[i].tw = tw; twv[i].tv = tv;
}
double find(struct ele *a, int n)
{  int i, k, f;
   double maxv, tw, tv, totv;
   maxv=0;
   for(totv=0.0, k=0; k<n; k++) totv += a[k].value;
   next(0, 0.0, totv);  i = 0;
   while(i >=0){
         f = twv[i].flg;  tw = twv[i].tw;  tv = twv[i].tv;
         switch(f){
          case 1:  twv[i].flg++;                  /*先考虑被选中*/
               if(tw+a[i].weight <= limitW)       /*选中可行吗？*/
                 if(i< n-1){                      /*后面还有物品吗？*/
                   next(i+1, tw+a[i].weight, tv);  /*置 I+1 物品的状态*/
                   i++;
                 }
                 else{                              /*是一个更好的侯选解*/
                   maxv = tv;
                   for(k=0; k<n; k++)
                      cop[k] = twv[k].flg !=0;
                 }
                 break;
         case 0: i--; break;                      /*回退*/
         default:                                 /*f == 2*/
                   twv[i].flg = 0;
                   if(tv - a[i].value > maxv)      /*不选 i 号物品可行吗？*/
                     if(i< n-1){                   /*后面还有物品吗？*/
                        next(i+1, tw, tv-a[i].value);
                        i++;
                     }
                     else{
                       maxv = tv-a[i].value;
                       for(k=0; k<n; k++)
                       cop[k] = twv[k].flg !=0;
```

```
                    }
                  brealk;
            }
    }
    return maxv;
}

void main()
{    double maxv;
     printf("输入物品种数. "); scanf("%d", &n);
     printf("输入限制重量. "); scanf("%lf", &limitW);
     printf("输入各物品的重量和价值. \n");
     for(k=0; k<n; k++)
     scanf("%lf%lf", &a[k].weight, &a[k].value);
     maxv = find(a, n);
     printf("\n选中的物品为\n");
     for(k=0; k<n; k++)
      if(cop[k]) printf("%4d", k);
     printf("\n总价值为%lf\n", maxv);
}
```

9.5 回溯法

回溯法也称为试探法，该方法首先暂时放弃关于问题规模大小的限制，并将问题的侯选解按某种顺序逐一枚举和检验。当发现当前侯选解不可能是解时，就选择下一个侯选解；倘若当前侯选解除了还不满足问题规模要求外，满足所有其他要求时，继续扩大当前侯选解的规模，并继续试探。如果当前侯选解满足包括问题规模在内的所有要求时，该侯选解就是问题的一个解。在回溯法中，放弃当前侯选解，寻找下一个侯选解的过程称为回溯。扩大当前侯选解的规模，并继续试探的过程称为向前试探。

【例 9. 6】在 9（3×3）个方格的方阵中要填入数字 1 到 N（N>=10）内的某 9 个数字，每个方格填一个整数，使所有相邻两个方格内的两个整数之和为质数。试求出所有满足这个要求的各种数字填法。

可用试探法找到问题的解，即从第一个方格开始，为当前方格寻找一个合理的整数填入，并在当前位置正确填入后，为下一个方格寻找可填入的合理整数。如不能为当前方格找到一个合理的可填整数，就要回退到前一方格，调整前一方格的填入数。当为第 9 格也填入合理的整数后，就找到了一个解，将该解输出，并调整第 9 格的填数去找下一个解。

为找到一个满足要求的 9 个数的填放，从还未填一个数开始，按某种顺序（如从小到大的顺序）每次在当前位置填入一个整数，然后检查当前填入的整数是否能满足要求。在满足要求的情况下，继续用同样的方法为下一方格填入整数。如果最近填入的整数不能满足要求，就改变填入的整数。如对当前方格试尽所有可能整数，都不能满足要求，就得回退到前一格，并调整前一格填入的整数。如此重复执行扩展、检查或调整、检查，直到找到一个满足问题要求的解，将解输出。

【例 9.6 算法 1】 回溯法找一个解的算法。

```
{ int m=0, ok=1;                          /*空状态是一个合理的情况*/
  int n =8;
  do{                                     /*重复执行以下操作*/
    if(ok)  扩展;
    else      调整;
    ok =检查前 m 个整数填放的合理行;
  }while(( !ok || m != n)&&(m !=0));
  if(m != 0) 输出解;                      /*找到了解*/
  else 输出无解报告;
}
```

如程序要找全部解，则在将找到的解输出后，应继续调整最后位置上填放的数，试图去找下一个解。相应的算法如下：

【例 9.6 算法 2】 回溯法找全部解的算法。

```
{  int m=0, ok=1;                         /*空状态是一个合理的情况*/
   int n =8;
   do{                                    /*重复执行以下操作*/
     if(ok)
      if( m ==n){
        输出解;
        调整;
      }
      else 扩展;
     else 调整;
     ok = 检查前 m 个整数填放的合理性;
   }while(m !=0);
}
```

为了确保程序能够终止，调整时，必须保证曾被放弃过的填数序列不会被再次试验，即要求按某种有序模型生成填数序列。给解的侯选者设定一个被检验的顺序，按这个顺序逐一形成侯选者并检验。从小到大或从大到小都是可以采用的办法。如扩展时，先在新位置填入整数 1，调整时，找当权侯选解中下一个还未被使用过的整数。将上述扩展、调整、检验都编写成函数，细节见以下找全部解的程序。

【例 9.6 程序】

```
#include <stdio.h>
#define N 12
void write(int a[])
{   int i, j;
    for(i=0; i<3; i++){
      for(j=0; j<3; j++);
          printf("%3d", a[3*i+j]);
      printf("\n");
```

```
    }
    scanf("%*c");
}
int b[N+1];
int a[10];
int isPrime(int m)
{    int i;
     int primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, -1};
     if(m == 1 || m %2 == 0) return 0;
     for(i=0; primes[i] >0; i++)
        if(m == primes[i]) return 1;
     for(i =3;  i*i <=m; ){
         if(m%i == 0) return 0;
         i+= 2;
     }
     return 1;
}
int checkMatrix[][3] = {{-1}, {0, -1}, {1, -1},
                {0, -1}, {1, 3, -1}, {2, 4, -1},
                {3, -1}, {4, 6, -1}, {5, 7, -1};
int selectNum(int start)
{   int j;
    for(j = start; j<=N; j++)
        if(b[j]) return j;
    return 0;
}
int check(int pos)
{   int i, j;
    if(pos < 0) return 0;
    for(i=0; (j = checkMatrix[pos][I] >=0; i++)
        if(! isPrimes(a[pos] + a[j]))
              return 0 ;
    return 1;
)
int extend(int pos)
{
  a[++pos] = selectNum(1);  b[a[pos]] = 0;
  return pos;
}

int change(int pos)
{    int j;
     while(pos >=0&&(j= selectNum(a[pos]+1)) ==0)
     b[a[pos--]] =1;
```

```
    if(pos < 0) return -1;
    b[a[pos]] =1;
    a[pos] =j;
    b[j] =0;
    return pos;
}
void find()
{   int ok=1, pos=0;
    a[pos]=1; b[a[pos]=0;
    do{
       if(ok)
         if(pos ==8){
           write(a);
           pos = change(pos);
         }
         else pos = extend(pos);
       else pos = change(pos);
       ok = check(pos);
    }while(pos >=0);
]
void main()
{  int t;
   for(i =1; i <=N; i++) b[i] =1;
   find();
}
```

【例 9.7】找出从自然数 1，2，…，n 中任取 r 个数的所有组合。例如 n=5，r=3，所有组合为：

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

采用回溯法找问题的解，将找到的组合以从小到大顺序存于 a[0]，a[1]，…a[r-1]中，组合的元素满足以下性质：

（1）a[i+1] > a[i]，后一个数字比前一个数字大；

（2）a[i] -i <= n-r+1。

按回溯算法思想，找解过程可以叙述如下：首先放弃组合数个数为 r 的条件，侯选组合从只有一个数字 1 开始。因该侯选解满足除问题规模之外的全部条件，扩大其规模，并使其满足上述条件（1），侯选组合改为 1，2。继续这一过程，得到侯选组合 1，2，3。该侯选解满足包括问题规模在内的全部要求，因而是一个解。在该解的基础上，选下一个侯选解，因 a[2]上的 3 调整为 4，以及以后调整为 5 都满足问题的全部要求，得到解 1，2，4 和 1，2，5。由于对 5 不能再作调整，就要从 a[2]回溯至 a[1]，这时的 a[1]=2，可以调整为 3，并向前试探，得到解 1，3，4。重复上述向前试探和向后回溯，直至要从 a[0]再回溯时，说明已找完问题的全部解。按上述思想写成程序如下：

【例 9.7 程序】

```
#include <stdio.h>
#define MAXN 100
int a[MAXN];
void comb_back(int m,  int r)
{   int i, j;
    i = 0;  a[i] =1;
    do{
      if(a[i]-i <=m-r+1){                  /*还可以向前试探*/
          if(i == r-1){                        /*已找到一个组合*/
              for(j=0; j<r; j++) printf("%4d", a[j]);
              printf("\n");
              a[i]++;
              continue;
          }
          i++;                                 /*向前试探*/
          a[i] = a[i-1] +1;
     }
     else{                                  /*回溯*/
          if(i ==0) return;                    /*已找到了全部解*/
          a[--i]++;
     }
    }while(1);
}

void main()
{comb_back(5, 3);
}
```

【例 9.8 】 求出在一个 n×n 的棋盘上，放置 n 个不能互相捕捉的国际象棋"皇后"的所有布局。

这是来源于国际象棋的一个问题。皇后可以沿着纵横和两条斜线 4 个方向互相捕捉。如图 9.3 所示，一个皇后放在棋盘第 4 行第 3 列位置上，则棋盘上凡打"*"号的位置上的皇后就能与第 4 行第 3 列位置上的皇后相互捕捉。

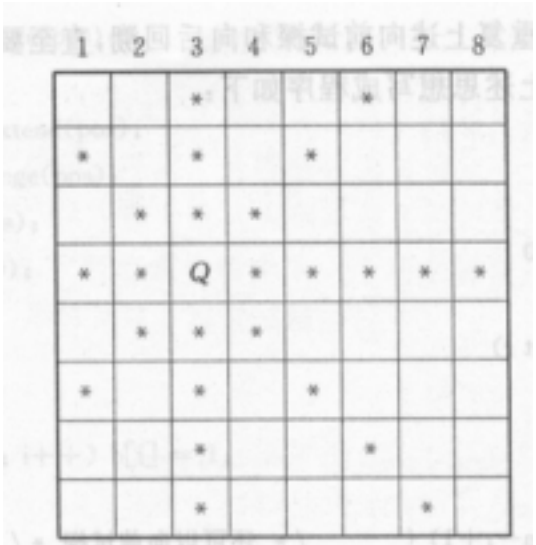


图 9.3 皇后相互捕捉位置示意图

从图 9.3 得到以下启示，一个合适的解应是在每列、每行上只有一个皇后，且在一条斜线上也只有一个皇后。

求解过程从空配置开始，在第 1 列至第 m 列为合理配置的基础上，再配置第 m+1 列。直至第 n 列配置也是合理时，就找到了一个解。接着改变第 n 列配置，希望获得下一个解。另外，在任何一列上，可能有 n 种配置。开始时，配置在第 1 行，以后改变时，顺次选择第 2 行、第 3 行、…、直到第 n 行。当第 n 行配置也找不到一个合理的配置时，就要回溯，去改变前一列的配置。得到求解皇后问题算法如下：

【算法】

```
{ 输入棋盘大小值 n；期                    /*一个 n*n 的棋盘*/
  m=0;                                   /*从空配置开始*/
  good=1;                                /*空配置中皇后不相互捕捉*/
  do{                                    /*找循环解*/
  if(good)
    if(m == n){                          /*找到了一个解*/
       输出解;
       改变之，形成下一个侯选解;
    }
    else 扩展当前侯选解至下一列;
  else 改变之，形成下一个侯选解;
  good = 检查当前侯选解的合理性;
  }while( m!=0);
}
```

在编写程序之前，先确定表示棋盘的数据结构。比较直观的方法是采用一个二维数组，但仔细考察就会发现，这种表示方法给调整侯选解及检查其合理性带来困难。更好的方法乃是尽可能直接表示那些常用信息。对于本题来说，"常用信息"并不是皇后的具体位置，而是"一个皇后是否已经在某行和某条斜线合理地安置好了"。因在每一列上恰好放一个皇后，引入一个一维数组（设数组名为 col），值 col[j]表示在棋盘第 j

列、col[j]行上有一个皇后。如 col[3]的值为 4，就表示在棋盘的第 3 列、第 4 行有一个皇后。另外，为了使程序在找完了全部解后回溯到最初位置，设定 col[0]的初值为 0。当回溯到第 0 列时，说明程序已求得全部解（或无解），结束程序执行。

为使程序在检查皇后配置的合理性方面简易方便，引入以下三个工作数组。

（1）数组 a[]，a[k]表示第 k 行上还没有皇后；

（2）数组 b[]，b[k]表示第 k 条右高左低斜线上没有皇后；

（3）数组 c[]，c[k]表示第 k 条左高右低斜线上没有皇后。

棋盘中同一右高左低斜线上的方格，他们的行号与列号之和均相同；同一左高右低斜线上的方格，它们的行号与列号之差均相同。

初始时，所有行和斜线上均没有皇后，从第 1 列的第 1 行配置第 1 个皇后开始；在第 m 列 col[m]行放置了一个合理的皇后后，准备考察第 m+1 列时，在数组 a[]，b[]，c[]中为第 m 列，col[m]行的位置设定有皇后标志；当从第 m 列回溯到第 m-1 列，并准备调整第 m-1 列的皇后配置时，清除在数组 a[]，b[]，c[]中设置的关于第 m-1 列，col[m-1]行有皇后的标志。一个皇后在 m 列，col[m]行方格内配置是合理的，由数组 a[]，b[]，c[]对应位置的值都为 1 来确定。细节见以下程序。

【例 9.8 程序 1】

```
#include <stdio.h>
#include <stdlib.h>
#define MAXN 20
int n, m, good;
int col[MAXN+1], a[MAXN+1];
int b[2*Maxn+1], c[2*MAXN+1];
void main()
{ int j;
  char awn;
  printf(" Enter n. ");  scanf("%d", &n);
  for(j=0; j <= n; j++) a[j] =1;
  for(j=0; j<=2*n; j++) b[j] = c[j] =1;
  m=1;  col[1] =1;  good=1;  col[0] =0;
  do{                                    /*找解循环*/
     if(good)
       if(m == 0){                       /*找到一个解*/
            printf("列\t 行");
            for(j=1; j<=n; j++)
                printf("%3d\t%d\n", j, col[j]);
            printf("Enter a character(Q/q for exit)! \n");
            scanf("%c", &awn);
            if(awn == 'Q' || awn == 'q')   exit(0);
```

```
            while(col[m] == n){
                m--;                        /*清除关于第 m-1 列，col[m-1]行有皇后标志*/
                a[col[m]] = b[m+col[m]] = c[n+m-col[m]] = 1;
            }
            col[m]++;                       /*调整第 m 列的皇后配置*/
       }
       else{                                /*在 m 列，col[m]行位置设定有皇后标志*/
            a[col[m]] = b[m+col[m]] = c[n+m-col[m]] = 0;
            col[++m] =1;                    /*第 m+1 列，从第 1 行开始配置*/
          }
     else{                                  /*回溯调整*/
          while(col[m] ==n){                /*回溯*/
            m--;                            /*清除关于第 m-1 列，col[m-1]行有皇后标志*/
            a[col[m]] = b[m+col[m]] = c[n+m-col[m]] = 1;
          }
          col[m]++;                         /*调整第 m 列的皇后配置*/
     }
     good = a[col[m]] && b[m+col[m]] &&c[n+m-col[m]];
  }while(m!=0);
}
```

试探法找解算法也常常被编写成递归函数，下面两程序中的函数 queen_all 和函数 queen_one 能分贝用来解皇后问题的全部解和一个解。

【例 9.8 程序 2】 递归方法找皇后问题全部解

```
#include <stdio.h>
#include <stdlib.h>
#define MAXN 20
int n;
int col[MAXN+1], a[MAXN+1];
int b[2×Maxn+1], c[2×MAXN+1];
void queen_all(int k,  int n)          /*在 n×n 棋盘的第 k 列上找合理的配置*/
{   int i, j;
    char awn;
    for(i=1; i <= n; i++)              /*第 k 列共有 n 种配置，分别在 1 至 n 行上*/
       if(a[i]&&b[K+i]&&c[n+k-i]){     /*测试合理性*/
         col[k] = i;
         a[i] =b[k+i]=c[n+k-i] =0;     /*置对应位置有皇后*/
         if(k==n){                     /*找到一个解*/
              printf("列\t 行");
              for(j=1; j<=n; j++)
                  printf("%3d\t%d\n", j, col[j]);
              printf("Enter a character(Q/q for exit)! \n");
              scanf("%c", &awn);
```

```
            if(awn == 'Q' || awn == 'q') exit(0);
         }
         else queen_all(k+1, n);  /*递归至第 k+1 列*/
         a[I] =b[k+I] =c[n+k-I] =1;  /*恢复对应位置无皇后*/
     }
}

void main()
{
    int j;
    printf("Enter n. \n");  scanf("%d", &n);
    for(j=0; j<=n; j++) a[j] =1;
    for(j=0; j<= 2*n; j++)b[j]=c[j]=1;
    queen_all(1, n);
}
```

采用递归方法找一个解与找全部解稍有不同，在找一个解的算法中，递归算法要对当前候选解最终是否能真正成为解要有回答。当它能成为最终解时，递归函数就不再递归试探，立即返回；若不能成为解，就得继续试探。设函数 queen_one 返回 1 表示找到解，返回 0 表示当前候选解不能成为解。细节见以下函数。

【例 9.8 函数】 递归方法找皇后问题一个解

```
#define MAXN 20
int n;
int col[MAXN+1], a[MAXN+1];
int b[2*Maxn+1], c[2*MAXN+1];
void queen_one(int k,  int n) /*在 n×n 棋盘的第 k 列上找合理的配置*/
{
    int I,ound;
    I = found = 0;
    while(!found && I <n){ /*未找到解，且还未试尽所有可能循环*/
        I++;
        if(a[I]&&b[K+I]&&c[n+k-I]){ /*测试合理性*/
            col[k] = i;
            a[I] =b[k+I]=c[n+k-I] =0;  /*置对应位置有皇后*/
            if(k==n) return 1;
            else found = queen_one(k+1, n);  /*递归至第 k+1 列，返回找到与否信息*/
            a[I] =b[k+I]=c[n+k-I] =1;
        }
    }
    return found;
}
```

9.6 贪婪法

贪婪法是一种不追求最优解，只希望得到较为满意解的方法。贪婪法一般可以快速得到满意的解，因为它省去了为找最优解要穷尽所有可能而必须消耗的大量时间。贪婪法常以当前情况为基础作最优选择，而不考虑各种可能的整体情况，所以贪婪法不要回溯。

例如平时购物找钱时，为使找回的零钱的硬币数最少，不考虑找零钱的所有各种方案，而是从最打面值的币种开始，按递减的顺序考虑各币种，先尽量用大面值的币种，当不足大面值币种的金额时才去考虑下一种较小面值的币种。这就是在采用贪婪法。这种方法在这里总是最优，是因为银行对其发行的硬币种类和硬币面值的巧妙安排。如只有面值为 1，5 和 11 单位的硬币，而希望找回总额为 15 单位的硬币。按贪婪算法，应找 1 个 11 单位面值的硬币和 4 个 1 单位面值的硬币，共找回 5 个硬币。但最优解答应是 3 个 5 单位面值的硬币。

【例 9.9】用贪婪法解装箱问题。

装箱问题可简述如下：设有编号为 0,1，…，n－1 的 n 种物品， 体积分别为 v0,v1,vn-1。将这 n 种物品装到容量都为 V 的若干箱子里。约定这 n 种物品的体积均不超过 V，即对于 0<=i<n 有 0<vi<V。不周的装箱方案所需要的数目可能不同，装箱问题要求使装尽这 n 种物品的箱子数要少。

若考察将 n 件物品的集合分划分成 n 个或小于 n 个物品的所有子集，最优解就可以找到，但所有可能分划的总数太大。对适当大的 n，找出所有可能的划分要花费的时间是无法承受的。为此，对装箱问题采用非常简单的近似算法，即贪婪法。该算法依次将物品放到它第一个能放进去的箱子中，该算法虽不能保证找到最优解，但还是能找到非常好的解。不失一般性，设 n 件物品的体积是按从大到小排好序的，即有 v0>=v1>=…>=vn-1。如不满足上述要求，只要先对这 n 件物品按它们的体积从大到小排序，然后按排序结果对物品重新编号即可。装箱算法简单描述如下：

【算法】 一个简单的装箱算法。

```
{   输入箱子的容积;
    输入物品种数 n;
    按体积从大到小排序，输入各物品的体积;
    预置已用箱子链为空;
    预置已用箱子计数器 box_count 为 0;
    for(i=0;i<n;i++){ /*物品按以下步骤装箱*/
        从已用的第一只箱子开始顺序寻找能放入物品的箱子;
        if(已用箱子都不能再放物品 i){
            另用一只箱子，并将物品 i 放入该箱子;
            box_count++;
        }
        else
            将物品放入箱子;
    }
}
```

上述算法能求出需要的箱子数 box_count，并能求出各箱子所装物品。下面的例子说明该算法不一定能找到最优解。设有 6 种物品，它们的体积分别为：60,45,35,20,20,20 单位体积，箱子的容量为 100 单位体积。按上述算法计算，需三只箱子，各箱子所装物品分别为：第一只箱子装物品 1,3；第二只箱子装物品 2,4,5；第三只箱子装物品 6。而最优解为两只箱子，如分别装物品 1,4,5 和 2,3,6 等。

若每只箱子所装物品用链表来表示，链表首结点指针存于一个结构中，结构记录尚剩余的空间量和该箱子所装物品链表的首指针。另将全部箱子的信息也构成链。以下是按以上算法编写的程序。

【例 9.9 程序】 用贪婪法解装箱问题。

```
#include <stdio.h>
#include <stdlib.h>
typedef struct ele{ /*物品结构信息*/
    int vno; /*物品号*/
    struct ele *link; /*另一物品的指针*/
}ELE;
typedef struct hnode{
    int remainder; /*箱子尚剩余空间  */
    ELE *head; /箱内物品链首元指针*/
    Struct hnode *next; /*箱子链的后继箱子指针*/
}HNODE;
void main()
{   int n,i,box_count,box_volume,*a;
    HNODE *box_h,*box_t,*j;
    ELE *p; *q;
    printf("输入箱子容积\n"); scanf("%d",&box_volume);
    printf("输入物品种数\n"); scanf("%d",&n);
    a = (int *)malloc(sizeof(int)*n); /*存储物品体积信息的数组*/
    printf("请按体积从大到小顺序输入各物品的体积:");
    for(i=0;i<n;i++) scanf("%d",a+i);
    box_h = box_t = NULL; /*预置已用箱子链为空*/
    box_count = 0; /*预置已用箱子计数器 box_count 为 0  */
    for(i=0;i<n;i++){ /*物品按以下步骤装箱*/
        /*从第一只箱子开始顺序寻找能放入物品 i 的箱子 j*/
        p = (ELE *)malloc(sizeof(ELE));
        p->vno = i;
        for(j=box_h;j !=NULL; j = j->next)
            if(j->remainder >= a[i]) break; /*找到还可装物品的箱子*/
            if(j == NULL){ /*已用箱子都不能  装物品*/
                j = (HNODE*)malloc(sizeof(HNODE)); /*使用一只新箱子*/
                j->remainder = box_volume -a[i];
                j->head = NULL;
                if(box_h == NULL) box_h = box_t =j;
                else box_t = box_t->next = j;
                j->next = NULL; box_count ++;
            }
```

```
            else
                j->remainder -= a[i]; /*将物品 i 放入箱子 j*/
            for(q=j->head;q!=NULL&&q-link!=NULL;q=q->link);
            if(q == NULL){ /*新启用的箱子*/
                q->link = j->head; j->head = p;
            }
            else{
                p->link = NULL; q->link = p;
                }
        }
        /*输出结果  */
        printf("共使用%d 只箱子.",box_count);
        printf("各箱子装物品情况如下：");
        for(j=box_h,i=1;j != NULL;j=j-next,i++){ /*第 i 只箱子情况*/
            printf("第%2d 只箱子，还剩余容积%4d，所装物品有：\n",I,j->remainder);
            for(p=j->head;p!=NULL;p=p->link)
                printf("%4d",p->vno+1);
            printf("\n");
    }
}
```

【例 9.10】 马的遍历问题。在 8×8 方格的棋盘上，从任意指定的方格出发，为马寻找一条走遍棋盘每一格并且只经过一次的一条路径。

马在某个方格，可以在一步内到达的不同位置最多有 8 个，如图 9.4 所示。如用二维数组 board[][]表示棋盘，其元素记录马经过该位置时的步骤号。另对马的 8 种可能走法（称为着法）设定一个顺序，如当前位置在棋盘的(i,j)方格，下一个可能位置依次为
(i+2,j+1),(i+1,j+2),(i-1,j+2),(i-2,j+1),(i-2,j-1),(i-1,j-2),(i+1,j-2),(i+2,j-1)，实际可以走的位置尽限于还未走过的和不越出棋盘边界的那些位置。为便于程序统一处理，可引入两个数组，分别存储各种可能走法对当前位置的纵横增量。



图 9.4 马走一步着法示意图

对于本题，一般可采用回溯法找解，这里采用 Warnsdoff 策略求解，这也是一种贪婪法，其选择下一出口的贪婪标准是在那些允许走的位置中，选择出口最少的那个位置。如马的当前位置(i,j)只有三个出口，它们是位置(i+2,j+1),(i-2,j+1)和(I-1,j-2)，如分别走到这些位置，这三个位置又分别会有不同的出口，假定这三个位置的出口个数分别为 4,2，3，则程序就选择让马走向(i-2,j+1)位置。

由于程序采用的是一贪婪法，整个找解过程是一直向前，没有回溯，所以能非常快地找到解。但是，对于某些开始位置，实际上有解，而该算法不能找到解。对于找不到解的情况，程序只要改变 8 种可能出口的选择顺序，就能找到解。改变出口选择顺序，就是改变有相同出口时的选择标准。以下程序考虑到这种情况，引入变量 start，用于控制 8 种可能着法的选择顺序。开始时为 0，当不能找到解时，就让 start 增 1，重新找解，细节见以下程序。

【例 9.10 程序】

```
#include <stdio.h>
int deltai[] ={2,1,-1,-2,-2,-1,1,2};
int deltaj[] ={1,2,2,1,-1,-2,-2,-1};
int board[8][8];
int exitn(int I,int j, int s, int a[])
{         /*求(I,j)的出口数，和各出口号于 a[ ]，a 是顺序选择着法的开始序号*/
    int i1,j1,k,count;
    for(count=k=0;k<8;k++){
    i1 = i+deltai[(s+k)%8];
    j1 = j+deltaj[s+k]%8};
    if(i1>=0&&i1<8&&j1>=0&&j1<8&&board[i1][j1]==0)
        a[count++] = (s+k)%8;
    }
    return count;
}
int next(int i,int j, int s) /*选下一出口，s 是顺序选择着法的开始序号*/
{   int m,k,kk,min,a[8],b[8],temp;
    m = exitn(i,j,s,a); /*确定(I,j)的出口个数  */
    if(m==0) return -1; /*没有出口*/
    for(min=9,k=0;k<m;k++){ /*逐一考察各出口*/
        temp = exitn(i+delatai[a[k]],j+deltaj[a[k]],s,b);
        if(temp < min){ /*找出有最少出口数的出口*/
            min = temp;
            kk = a[k];
        }
    }
    return kk; /*返回选中的着法*/
}
void main()
{   int sx,sy,i,j,step,no,start;
    for(sx=0;sx<8;sx++)
```

```
    for(sy=0;sy<8;sy++){
        start=0; /*从 0 号着法开始顺序检查*/
        do{
            for(i=0;i<8;i++)
                for(j=0;j<8;j++) board[I][j]=0; /*清棋盘*/
            board[sx][sy]=1;
            i=sx; j=sy'
            for(step=2;step<=64;step++){
                if((no=next(i,j,start) == -1) break;
                i += deltai[no]; /*前进一步*/
                j += deltaj[no];
                Board[I][j] = step;
            )
            if(step > 64) break;
            start++; /*最先检查的着法序号增 1*/
        }while(step <=64);
        for(i=0;i<8;i++){
            for(j=0;j<8;j++)
                printf("%4d",board[i][j]);
            printf("\n\n");
        }
        scanf("%*c"); /*键入回车，找下一个起点的解*/
    }
}
```

9.7 分治法

分治法也许是最广泛使用的算法设计方法。其基本思想是把大问题分解分一些较小的问题，然后由小问题的解方便地构造出大问题的解。在介绍递归算法时，读者已看到了这样的应用。为用实例说明该方法，考虑著名的 Hanoi 问题。有 A,B,C 三根针，和一些中间有孔大小不等的金片。初始时，金片全在 A 针上，其中最大的在下面，从大到小，最小的在最上面。要求将 A 针上的金片全移至 B 针。限制每次只移一张金片，并且不允许将大的金片放在小金片的上面，但移动时可以将金片放在 C 针上。问题要求给出金片的移动过程。

移动 A 针上最上面 n 张金片到 B 针的算法可采用分治法，由以下三个步骤组成：

（1） 将 A 针最上面的 n-1 张金片移至 C 针；

（2） 将现在露在 A 针最上面的第 n 张金片移至 B 针；

（3） 将暂存于 C 针上的 n-1 张金片移至 B 针。

移 n 张金片可采用递归实现，虽实际移动是一张张金片的移动，但按分治法思想，把上面的 n-1 张金片的移动与最下面的第 n 张金片的移动两件事分开，从而将 n 张金片的移动转换成更少张金片的移动，并且不

必考虑各针下面金片的情况。从概念上来理解，算法的正确性是非常明显的。例子说明分治法是一种很重要的方法，并且在大多数情况下，还是一个很有效的方法。

【例 9．11】 参加网球比赛的选手安排比赛日程。

设有 n(=2 的 k 次幂)位选手参加网球循环赛，循环赛共进行 n-1 天，每位选手要与其他 n-1 位选手赛一场，不轮空。试按此要求为比赛安排日程。

设 n 位选手被顺序编号为 1,2，…，n。比赛的日程表是一个 n 行 n-1 列的表，i 行 j 列的内容是第 i 号选手第 j 天的比赛对手。用分治法设计日程表，、就是从其中一半选手（2m-1 位）的比赛日程，导出全体（2m 位）选手的比赛日程。从众所周知的只有两位选手的比赛日程出发，反复这个过程，直至为 n 位选手安排好比赛日程为止。

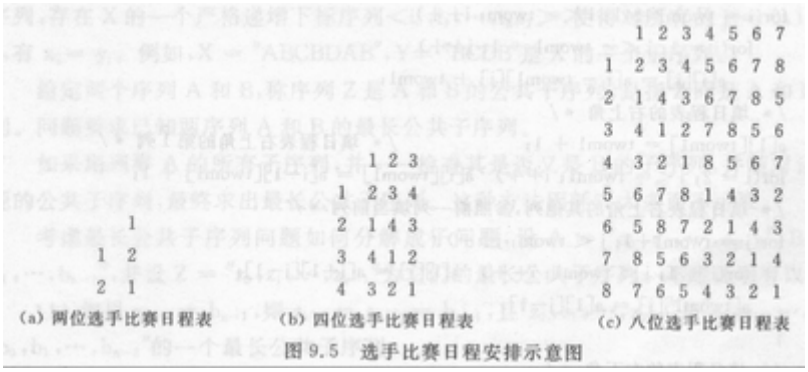


图 9.5 选手比赛日程安排示意图

为了从 2^{m-1}位选手的比赛日程表，导出 2m 位选手的比赛日程表，假定只有 8 位选手（见图 9.5），若 1 至 4 号选手之间的比赛日程填在日程表的左上角（4 行 3 列），5 至 8 号选手之间的比赛日程可填在日程表的左下角（4 行 3 列），而左下角的内容可由左上角对应项加上数 4 得到。至此，剩下的右上角（4 行 4 列）是为编号小的 1 至 4 号选手与编号大的 5 至 8 号选手之间的比赛安排日程。如在第 4 天，让 1 至 4 号选手分别与 5 至 8 号选手比赛。以后各天，依次由前一天的日程安排，让 5 至 8 号选手循环轮转即可。最后，参照右上角得到右下角的比赛日程，如图 9.5 所示。由以上分析，不难得到为 2k 位选手安排比赛日程的算法思路。细节见以下程序。

【例 9．11 程序】

```
#include <stdio.h>
#define MAXN 64
int a[MAXN+1][MAXN];
void main()
{   int twoml,twom,I,j,m,k;
    printf("指定（＝2 的 k 次幂）位选手，请输入 k. \n",&k);
    scanf("%d",&k);
    a[1][1]=2;
    a[2][1] =1; /*预设 1 两位选手的比赛日程*/
    m=1;twoml=1;
    while(m<k){
        m++;
```

```
        twom1 += twom1; twom = 2*twom1; /*为 2m 位选手安排比赛日程*/
        /*填日程表的左下角*/
        for(i=twom1+1;i<twom;i++)
            for(j=1;j<=twom1-1;j++)
                a[i][j] = a[i-twom1][j] + twom1;
            /*填日程表的右上角*/
        a[1][twom1] = twom1+1; /*填日程表右上角的第 1 列*/
        for(i=2;i<=twom1;i++) a[i][twom1] = a[i-1][twom1]+1;
        /*填日程表右上角的其他列，参照前一列填当前列*/
        for(j=twom1+1;j<twom;j++){
            for(i=1;i<twom1;i++) a[i][j] = a[i+1][j-1];
                a[twom1][j] = a[1][j-1];
        }
        /*填日程表的右下角*/
        for(j=twom1;j<twom;j++){
            for(i=1;i<twom1;i++)
                a[a[i][j]][j] = i;
        for(i=1;i<=twom;i++)
            for(j=1;j<twom1;j++) printf("%4d",a[i][j]);
            printf("\n");
        }
        printf("\n");
    }
}
```

9.8 动态规划法

经常会遇到复杂问题不能简单地分解成几个子问题，而会分解出一系列的子问题，简单有采用把大问题分解成子问题，并综合子问题的解导出大问题解的方法，问题求解耗时会按问题规模呈幂级数增加。

为了节约重复求相同问题的时间，引入一个数组，不管它们是否对最终解有用，把所有子问题的解存于该数组中，这就是动态规划法所采用的基本办法。以下用实例说明动态规划方法的用法。

【例 9．12】求用两字符序列的最长公共字符子序列。

这里所说的字符序列的子序列可能与读者理解的不同。这里的字符子序列是指从给定字符序列中随意地（不一定是连续一段）去掉若干字符（可能一个也不去掉）后所形成的字符列。令给定的字符序列 X="x0,x1,…,xm-1"，序列 Y="y0,y1,…,yk-1"是 X 的子序列，存在 X 的一个严格递增下标序列 <i0,i1,…,ik-1>，使得对所有的 j=0,1,…k-1，有 xij=yj。例如，X="ABCBDAB"，Y="BCDB"是的一个子序列。

给定两个序列 A 和 B，称序列 Z 是 A 和 B 的公共子序列，是指 Z 同是 A 和 B 的子序列。问题要求已知两序列 A 和 B 的最长公共子序列。

如采用列举 A 的所有子序列，并一一检查其是否又是 B 的子序列，并随时记录所发现的公共子序列，最终求出最长公共子序列。这种方法因耗时太多而不可取。

考虑最长公共子序列问题如何分解成子问题，设 A="a0,a1,…,am-1"，B="b0,b1,…,bn-1"，并设 Z="z0,z1,…zk-1"为它们的最长公共子序列。不难证明有以下性质：

(1) 如果 am-1=bn-1，则 zk-1=am-1=bn-1 且"z0,z1,…zk-2"是"a0,a1,…am-2"和"b0,b1,…bn-2"的一个最长公共子序列；

(2) 如果 am-1!=bn-1，则若 zk-1!=am-1，蕴含"z0,z1,…,zk-1"是"a0,a1,…,am-2"和"b0,b1,…bn-1 的一个最长公共子序列。

(3) 如果 am-1!=bn-1，则若 zk-1!=bn-1，蕴含"z0,z1,…,zk-1"是"a0,a1,…,zm-1"和"b0,b1,…,bn-1"的一个最长公共子序列。

这样，在找 A 和 B 的公共子序列时，如有 am-1=bn-1，则进一步解决一个子问题，找"a0,a1,…,am-2"和"b0,b1,bn-2"的一个最长公共子序列；如果 am-1!=bn-1，则要解决两个子问题，找出"a0,a1,…,am-2"和"b0,b1,bn-1"的一个最长公共子序列，和找出"a0,a1,…,am-1"和"b0,b1,bn-2"的一个最长公共子序列，再取两者中的较长者作为 A 和 B 的最长公共子序列。

定义 c[i][j]为序列"a0,a1,…,ai-1"和"b0,b1,…,bj-1"和的最长公共子序列的长度，计算 c[i][j]可递归地表述如下：

c[i][j]=0，如果 i=0，或 j=0；

c[i][j]=c[i-1][j-1]+1，如果 i,j>0，且 a[i-1]==b[j-1]

c[i][j] = max(c[i][j-1],c[i-1][j])，如果 i,j>0，且 a[i-1]!=b[j-1]

按此算式可写出计算两个序列最长公共子序的长度函数，见例 9.14 程序的函数 lcs_len。

由 c[i][j]的产生仅依赖于 c[i-1][j-1]、c[i-1][j]和 c[i][j-1]可知，利用函数 lcs_len()所产生的数组 c[ ][ ]，可以从 c[m][n]开始，跟踪 c[i][j]的产生过程，逆向构造出最长公共子序列。

【例 9．12 程序】

```
#include <stdio.h>
#include <string.h>
#define N 100
char a[N],b[N],str[N];
/*计算两个序列最长公共子序的长度函数*/
int lcs_len(char *a, char *b, int c[][N])
{   int m = strlen(a),n = strlen(b),i,j;
    for(i=0;i<=m;i++) c[i][0]=0;
    for(j=1;j<=n;j++) c[0][j] = 0;
    for(i=1;i<=m;i++)
```

```
        for(j=1;j<=n;j++)
            if(a[i-1]==b[j-1] c[i][j]=c[i-1][j-1]+1;
            else if(c[i-1][j] >= c[i][j-1])
                c[i][j] = c[i-1][j];
            else c[i][j] = c[i][j-1];
    return c[m][n];
)
/*构造最长公共子序列函数*/
char *build_lcs(char s[],char *a,char *b)
{   int k,i=strlen(a),j=strlen(b),c[N][N];
    k=lcs_len(a,b,c);
    s[k]='\0';
    while(k>0)
        if(c[i][j] == c[i-1][j]) i--;
        else if(c[i][j] == c[i][j-1]) j--;
        else{ s[--k] = a[i-1];
              i--;j--;
            }
    return s;
}
void main()
{   printf("Enter two string(<%d)!\n",N);
    scanf("%s%s",a,b);
    printf("LCS = %s\n",build_lcs(str,a,b));
}
```

【例 9. 13】 凸多边形的最优三角剖分。

一个由 n 个顶点构成的凸多边形 P 可用按顺时针方向列出其所有顶点来表示，如图 9.6 的凸多边形可表示成 P=<v0,v1,v2,v3,v4,v5,v6>。给定两个不相邻的顶点 vp 和 vq，线段<vp,vq>为多边形的弦。多边形的最优三角剖分是指已知凸多边形 P=<v0,v1,…,vn-1>，要求给出一种三角剖分方案，使得剖分所用的纺、弦长之和为最少。

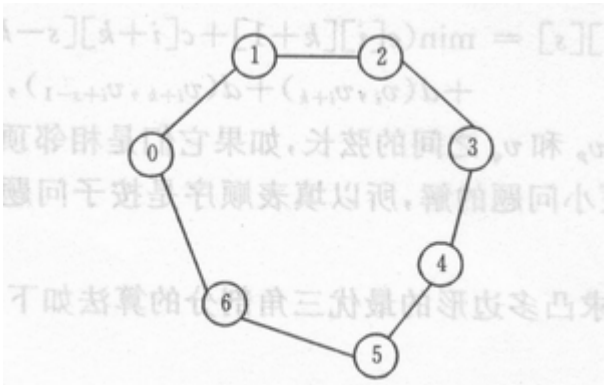


图 9.6 一个由 7 个顶点组成的凸多边形

考虑一个 n 个顶点的凸多边形的一种最优三角剖分，如该最优剖分包含三角形△v0vkvn-1，0<k<n-1。该三角剖分的最优值由三角形△v0vkvn-1 的弦长值和与对两个多边形<v0,v1,…vk>和<vk,vk+1,…vn-1>的三角剖分中最优值之和。不难看出，最优三角剖分应是对所有可能的 k 取其最小值。

现定义 Si,s 表示自顶点 i 开始，并由个 s 顶点组成的多边形<vi,vi+1,…vi+s-1>。为求解 Si,s，必须考虑以下三种可能形式的选择，则图 9.7。

（1） 选顶点 vi+s-2，构成△vivi+s-2vi+s-1 和子问题 Si,s-1。

（2） 选顶点 vi+1，构成△vivi+1vi+s-1 和子问题 Si+1,s-1。

（3） 对 2 和－3 之间的某个 k，选顶点 vi+k，构成△vivi+kvi+s-1 和两个子问题 Si,k+1 及 Si+k,s-k。

在这里，若约定当子问题的大小 s<3 时，值为 0；而当 s＝3 时其弦长之和确定，右把上述三种可能合而为一。即，对 1 和 s-2 之间的某个 k，解两个子问题 Si,k+1 及 Si+k,s-k。

对于三角剖分问题，如采用递归法求解，在递归过程中会反复地求同样的子问题，解题所费时间会随问题规模的大小呈幂欠次增加。为此采用动态规划方法，记子问题 Si,s 的最优三角剖分值为 c[i][s]，对于 s>=4，c[i][s]的计算公式为：

$c[i][s] = \min(c[i][k+1]+c[i+k][s-k]+d(v_i,v_{i+k})+d(v_{i+k},v_{i+s-1}),\ 1<=k<=s-2$

这里的 d(vp,vq)是顶点 vp 和 vq 之间的弦长，如果它们是相邻顶点，则其值约定为 0。由于给定问题的解依赖于更小问题的解，所以填表顺序是按子问题规模从小到大的顺序填写。

根据以上分析，得到求凸多边形的最优三角剖分的算法如下：

【例 9.13 算法】

```
{   边形的顶点个数 n;
    按逆时针顺序输入各顶点的坐标;
    计算各弦的长度填于表 d[ ][ ]; /*其中相邻顶点的长度为 0*/
    for(i0;in;i+) c[i[2]=0.0;
    for(i0;in;i+){ /*计算 c[i[3]，并形成 Si,3 的弦链表*/
        计算 c[i[3];
        形成 Si,3 的弦链表;
    ]
    for(s=4,s<=n;s++)
    for(i=0;i<n;i++){ /*计算 c[i][s]*/
        for(t=MAX,kk=1;kk <=s-2;kk++){
            temp=c[i][(kk+1)%n]+c[(i+kk)%n][(n+s-kk)%n]+
            d[i][(i+kk)%n]+d[(i+kk)%n][(i+s-1)%n];
        if(temp <t){ /*调整*/
            k=kk;t=temp;
        }
```

```
    }
    c[i][s] = temp;
    将弦<vi,vi+k>及 Si,k+1 和 Si+k,s-k 的剖分弦链接在一起;
    if(s==n) break; /*计算结束，最优值在 c[0][n]*/
    }
    输出结果;
}
```

第 2 章 基本数据结构及操作

2.1 数据抽象与程序设计

2.1. 1 抽象数据类型

程序设计过程的核心抽象,抽象数据类型是对象的数学模型,它给出一类数据的值集,并确定在该值集上允许的操作.抽象数据类型的构造与算法设计是密切不可分的,两者互相影响,互相依赖.

抽象数据类型反映了对象中的各种关系,各种数据结构用来表示或实现抽象数据类型这里所说的实现是设计级的,结构化的数据值是可分的,各成份值之间的关系反映了数学模型内的各种关系.

必须区分数据结构与程序设计语言中的数据类型这两个概念.语言中提供的基本数据类型以及供程序员构造结构化数据类型的方法是在编码级上实现数据结构的工具.

数据结构与语言中的类型设施有关,但两者并非互相对应的.一些最基本的数据结构,如记录、数组、字符串等在很多情况下可以直接利用语言提供的类型设施来实现,所谓"直接"是指语言本身提供了对这些结构的描述手段并且提供了对它们的操作.例如从一个记录值中取一个成份值,有许多常用的数据结构,如栈、队列、链表、树等。在很多语言中并没有相应的数据类型设施，需采用构造抽象数据类型的方法来实现它们。

以上简单叙这了在计算机求解问题的过程中，在总是描述（建立模型）、设计以及编码等各层次上抽象与实现的关系。在这个过程中,数据结构起着重要的作用。

本章中主要讨论一些最常用的数据结构,在程序中实现这些结构也限于经较简单的静态方法。

2．1．2 抽象数据类型的实现

最简单的数据组织方式之一是把若干个内部结构可能相同也可能不同的数据组合为一个整体,这就是记录结构,可以为此为例来说明实现抽象数据类型的方法。

[例 2．1]实现为理数的精确运算

有理数是最常用的数学模型之一。一般在计算机中用浮点数近似运算，本例中要求以分数形式进行精确的四则运算。

[解]分数是由分子和分母两个整数构成的结构化数据，可以很自然地用记录结构表示。

程序要求用户两次输入 4 个整数，分别作为两个分数的分子与分母，运算种类（加、减、乘、除四种运算之一）由用户选择。最后输出运算结果，分母若为 1 则不输出分母。

为简单起见，假设用户输入格式正确。

以下为用 C 语言写的程序：

```
/*有理数四则运算 /*
# include<stdio.h>
struct fraction /* 分数结构，由分子、分母构成 */
{
int numer,denom;
};
main()
{
struct fraction fract1,fract2,result;
int num ,den ,op=0;
static char opsym□="+ - *"/* 运算符表，供输出用 */
void make_fraction(),reduce(),add();
extern void subs(),mult(),div();
den=0;/* 输入第一个分数的分子、分母（若输入为 0，要求重输）

while(den==0)
{
printf("Enter first fraction.(numer denom (/=0))\n");
scanf("%d%d",&num,&den);
}
make_fraction(num,den,&fract1);/* 构造第一个分数 */
den=0;/* 输入第二个分数的分子、分母（若输入为 0，要求重输）
while(den==0)
{
printf("Enter second fraction.(number,dennom(/=0))\n");
scanf("%d%d",&num,&den);
}
make_fraction(num,den,&fract2);/* 构造第二个分数 */
while ((op<1)‖(op>4)/* 输入运算选择(不在指定范围内,要求重输) */
```

```
{
printf("select operation:+(1),-(2),*(3),/(4)");
scanf("%d",&op):
}
switch(op)/* 根据所指定的运算转相应予程序 */
{
case 1:add(fract1,fract2,&result);
break;
case 2:subs(fract1,fract2,&result);
break
case 3:mult(fract1,fract2,&result);
break
case 4:div(fract1,fract2,&result);

if(result.denom==1) /* 输出结果 */
printf("(%d%d)%c(%d%d)=%d\n",fract1.numer,)
fract1.denom,opsym[op-1],fract2.numer,
fract2.denom,result.numer);
else printf("(%d/%d)%c(%d/%d)=(%d/%d)\n",fractl.numer,
fract1.denom,opsym[op-1],fract2.numer,
fract2.denom,result.numer,result.denom);
}
/*函数 make_fraction 由两整数构造一个分数结构*/
void make_fraction(number1,number2,result_addr)
int number1,number2;
struct' frction *result_addr;
{
result_addr->numer=number1;
result-addr->denom=number2;
}
/* 函数 gcd 计算两个整数的最大公约数 */
int gud(x,y)
int x,y;
{
int m,n,r;
m=x,n=y,r=m%n;
}
return(n);
}
/* 函数 1cm 计算两个整数的最小公倍数 */
int lcm(x,y)
int x,y;
{
return(x*y/gcd(x,y));
```

```
}
/* 函数 reduce 将一分数约分为最简分数 */
void reducd(f1);
{
int;templ;
templ=gcd(f1->numer,f1->denom);
f1->numer=f1->number/templ;
f1->denom=f1->denom/templ;
}
/* 函数 add 计算两个分数的和 */
void add(f1,f2,result_addr)
struct fraction f1,f2;
struct fraction * result_addr;
{
result_addr->denom=lcm(f1.denom,f2.dencom);
result_addr->numer=f1.numer * (result_addr->denom/f1.denom)
+f2.numer * (result_addr->denom/f2.denom);
reduce(result_addr);
}
```

在上面的程序中，略去了计算分数差、积和商的三个函数 subs、mult 与 div，读者很容易自行写出。

2．1．3 指针

指针是数据位置的抽象。指针本身并不是一种结构型数据，它在实现名种数据结构时有广泛的应用，其最多重要的用途之一是构造各种动态数据结构，在本章中只讨论指针的非动态应用。

大多数常用语言中提供了指针类型，除指针外，用数据名与整型数也能实现抽象地址控制。

在[例 2.2]将复数从直角坐标表示转换为极坐标表示。

[解] 复数在工程计算中有广泛的应用，复数 a +bi 可以用二维实空间中的点坐标表示。在直角

坐标系中为一对有序实数(a,b),在极坐标中则为(r, θ),其中 , θ 为[0° ,360°]范围中的角度，满足等式：sin θ =b/r;cos θ =a/r。程序中 θ 用弧度表示，输出时转换为度。

```
/* 复数直角坐标表示转换坐标表示 */
# include<stdio.h>
# include<math.h>
```

```
# define pi 3.14159265
struct rectan/* 直角坐标 */
{
double real,imag;
};
struct polar/* 极坐标 */
{
double mag,angle;
}result;
main()
{
static strct rectan zr={10.0,10.0};
void to_polar();
to_polar(&zr,&result);
printf("\n magnitud=%f,angle=%f degress\n",
result.mag,result.angle);
{
/*函数 to-polar 求极坐标值 */
void to_polar(z1,z2)
struct rectan * z1;
struct polar * z2;
{
z2->mag=sprtl(z1->real * z1->real+z1->imag*z1->imag);
z2->angle=atan2(z1->image,z1->real)* 180.0/pi;
}
```

为简单起见,程序中没有输入功能,给变量 zr 指定了初值.注意,程序中利用 tg θ =b/a 求角度, θ 所在象限并不能由 tg θ 值唯一确定,还必须考虑 a、b 符号.这里利用了 C 语言数字库存函数 atan2.

应当指出,许多语言中对实参的值提供了保护,如 PASCAL 语言中规定可改变参数值的情况必须显式指出(即 VAR 参数段).而 C 语言中则规定不能改变实在参数的值,因此在本例中,不能直接用 result 作函数 to_polar 的参数,在函数中必须对 result 赋值,这里就必须用指向 result 的指针作为实在参数.

在下面的例了中,利用记录结构与指针实现字位操作.

[例 2. 3]生成字符二进制码。编一个程序，读入一串字符，逐行输出每个字符以及用八位二进数字表示的该字符的 ASCII 码;最高位为奇偶位,若该字符 ASCII 码中"1"出现奇数次则最高位为 1，否则最高位为 0。

[解]为了能将字符分解为字位，把一个字符看作由 8 个字位构成的记录结构数据，通过记录指针对其进行操作。C 语言程序如下：

```
/*输出字符的 ASCII 码，并加上奇偶校验位 */
#include<stdio.h>
struct bit
{
unsigned b 0:1,b1:1,b2:1,b3:1,b4:1,b5:1,b6:1,b7:1;
} *sp;/ * 把字符定义为记录结构 */
main()
{
int bit_sum,parity;
char string[21], * strptr;
while(1) / * 循环直到用户输入一个空行时中止 * /
printf("Enter a line of up to 20 characters. \n");
if(! * (strptr=gets(string)))
break;
while(* strptr !='\0') / * 处理一行 */
{
sp=(struct bit *)strptr;
bit_sum=sp_>b0+sp->b1+sp->b2+sp->b3+sp->b4+sp->b5+sp->b6;
partiy=bit_sum % 2;/ *计算奇偶位 */
printf("%c",* strptr);
printf("%d%d%d%d%d%d%d%d\n" ,parity ,sp->b6,sp->b5,sp->b4,sp->b3,sp->
b2,sp->b2,sp->b0);
strptr++;
}
}
}
```

采用本例中的技术,利用指针可以将由字位域组合成的记录结构用到已定义为其它类型的变量或一存储区域上.

2.2 数组

2.2. 1 数组的特征与表示

数组是最常用的数据结构之一,常用它来实现向量与矩阵.绝大多数常用语言都提供了数组类型设施,在有些语言中这甚至是唯一的结构类型.

数组由同种类型的数据构造而成,可以用某种有序类型值和为索引,随机地访问数组中的元素.一般均用整型数作为索引值,即数组下标值.

语言中普遍提供的计数循环控制机制正是考虑到数组结构特征而设置的,因此用数组结构处理多个同类型数据极为方便.

下面这个简单的例子描述了数组结构最常用的使用方式.

[例 2.4]用 Cramer 方法解实系数二元线性方程组.

[解]待解方程组形如:

$$a_{11}x_1 + a_{12}x_2 = a_{13}$$
$$a_{21}x_1 + a_{22}x_2 = a_{23}$$

Cramer 方法是计算下列三个行列式的值:

$$d_0=\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \quad d_1=\begin{vmatrix} a_{13} & a_{12} \\ a_{23} & a_{22} \end{vmatrix} \quad d_2=\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix}$$

原方程组有唯一组解当且仅当 d0 ≠0,此时:

x1=d1/d0,x2=d2/d0

用一个 2×3 的实型数组存放方程组系数(包括常数项).

C 语言程序如下:

```
/ * 用 Cramer 方法解二元线性方程组 */
#include<stdio.h>
main()
{
float a[2][3],d,x1,x2;
int I ,j;
for(I=1;I<2;I++)/ * 输入各系数 */
for(j=1;j<3;j++)
{
printf("Enter coef A[%d][%d]:"I,j);
scanf("%f",&a[I-1][j-1]);
}
d=a[0][0] *a[1][1]-a[0][1]*[1][0];
if(d==0) / * 无唯一解 * /
printf("\n No unique solution.\n");
else
{
printf("\n x1=%f\n,(a[0][2]*a[1][1]-a[0][1]*a[1][2]/d);
printf("\n x2=%f\n,(a[0][0]*a[1][2]-a[0][2]*a[1][0]/d);
}
}
```

Cramer 方法用于含三个以上方程的方程组很不方便,下面介绍的 Gauss_Jordan 方法也适用于较大的方程组。

[例 2．5]用 Gauss_Jordan 方法解线性实系数方程组。

[解]Causs_Jordan 方法的基本思想与用消元法手工解方程组相同。即逐个利用第 I（I=1,2,…， n）个方程消去其它 n-1 个方程中的第 I 个未知数。

例如要解议程组：

$2x_1+3x_2+5x_3=10$ ①
$x_1+2x_2+4x_3=5$ ②
$4x_1+7x_2+3x_3=15$ ③

分别将①乘（-0.5）加到②上,以及将①乘(-2)加到③上可消去②、③中的 x1:

$2x_1+3x_2+5x_3=10$ ①
$0.5x_2+1.5x_3=0$ ②
$x_2-7x_3=-5$ ③

再分别将②乘（-6）加到①上，以及②乘（-2）加到③上可消去①、③中的 x2：

$2x_1-4x_3=10$ ①
$0.5x_2+1.5x_3=0$ ②
$-10x_3=-5$ ③

以此类推，可以进行到每个方程中含一个未知数，便可用简单除法得到解。

与上例一样，用二维数组存放方程组中所有系数。假定方程组最多含 9 个方程，实际方程个数由用户输入指定。按上面所说的方法消元，结果使数组中除第 n+1 列（常数项）外，非零元素仅出现在主对角线上，即 aii。

循环消元的每一轮中，利用第 I 个方程消去第 j（j=1,2,…，n+1）个方程中的 xi 时要将诸 aik(k=1,2,…,n+1)均乘因子 aji/aii，因此要避免 aii 为 0，同时因机器中通常用浮点方式实现实型数，考虑误差也就避免 aii 的绝对值相对太小，所以在每轮消元处理前通过排序使 aii 位置上的值总是
ai.iai,I+1，….ai.k 中绝对值最大的。

算法流程图如图 2.1 所示,数组元素表示为 aj,I。

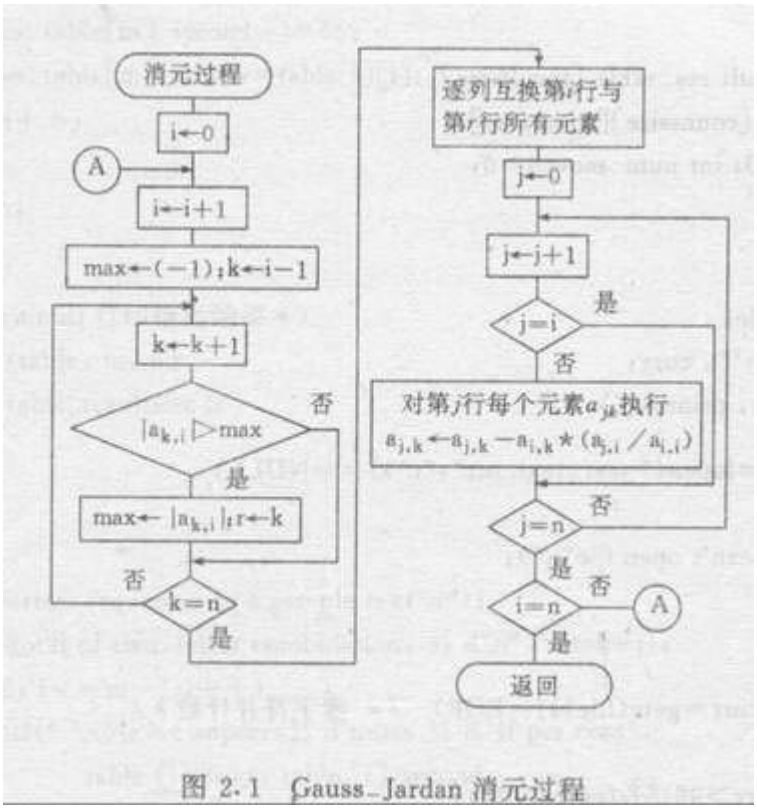


图 2.1 Gauss_Jardan 消元过程

在数据处理中最常见的结构之一是以记录结构为元素的数组。这种结构适于用来设计高效率处理大量数据的算法，也是建立动态结构的基础。下面的例子反映了这种结构的一种简单应用。

图 2.1Gauss-Jardan 消元过程

[例 2．6]字母组合统计。检查一段正文，计算基中各种双字母（指 a,b,c…，z，不分大小写）组合出现的次数以及各自在双字母组合总数中所占百分比；打印输出百分比值超过指定值（下面以 1%为例）的所有组合及其百分比，输出顺序按照其出按照其出现频度的降序。

[解]建立一个二维数组 count，其行、列号对应于一字母组合的第一、二个字母在字母表中的序号，其元素值即为相应字母组合已出现次数。如 count（0，1）=5 则表明字母组合"AB"已出现 5 次。

一维数组 res_take 则存放将要打印输出的结果，其元素为记录，结构如下：

第一字符	第二字符	出现次数

res_table[i]

输出所需的排序操作在 res_table 中进行。

假设待查正文段在文件 test_text 中。C 语言程序如下：

```
/ * 检检正文文件段中双字母组合出现率 */
#include <stdio.h>
#define countsize 26 / * 计数数组大小 * /
#define resultsize 100 / * 输出结果存放表大小 * /
#define threshold 1.0 / * 输出组合最小出现百分率 * /
struct result / * 输出结果表元素结构 * /
{
char first ,second
int occurs;
}
static struct result res_table[resultsize];
static int count[countsize][countsize];
long totalseq=0;int num_moved=0;
main()
{
FILE * file;
Char prev='',curr;
Void sort(),printseq();
1f((file=fopen("test_text.txt","r"))==NULL)
{
printf("can't open file\n");
exit();
while ((courr=getc(file))!=EOF) / * 读字符并计数 */
{
if ((curr>96)&&(curr<123))
curr=curr-32; / * 小写字母转换大写 * /
if ((curr>64)&&(curr<91))
{
count[prev-65][curr-65]+=1;
totalseq++;
}
prev=curr;
}
if(totalseq==0)
printf("Date contains no seqs. \n");
num_moved=move(count);
sort(num_moved,res_table);
printseq(res_table,num_moved,totalseq);
}
/ * 函数 move 将满足打印要求的组合送结果表并计数 * /
int move(table)
int talbe[countsize][countsize];
```

```
{
int n,m=o,I,j;
n=totalseq * threshold/100;
for(I=0;I<=25;I++)
for(j=0;j<=25;I++)
if(table[i][j]>n)
{
res_table[m].first=I+65;
res_table[m].second=j+65;
res_table[m].occurs=table[i][j];
m++;
}
return m;
}
/ * 函数 printseq 打印输出结果 * /
void printseq(table,m,n)
struct result table[resultsize];
int m,n;
{
int I;
printf("letter_sequences in a sample text\n");
printf("total of two_letter combination:%d\n",totalseq);
for(I=0;I<=m-1;I++)
printf("\n%c%c appears%d time % 8.4f per cent",table[i].first
table[i].second,
table[I].occurs * 100.0/n)
}
/ * 函数 sort 对 table 中 n 个元素按从大到小排序  *  /
void sort (n,table)
int n;
struct result table[resultsize];
{
int I,j;
struct result temp;
for(I=1;I<=n-2;j++)
if(table [j].occurs>table[i].occurs
{
temp.first=table[i].first;
temp.second=table[i].second;
temp.occurs=table[i].occurs;
table[i].first=table[j].first;
table[i].second=table[j].second;
table[i].occurs=table[j].occurs;
table[j].first=temp.first;
```

```
table[j].second=temp.second;
atbles[j].occurs=temp.occurs;
}
}
```

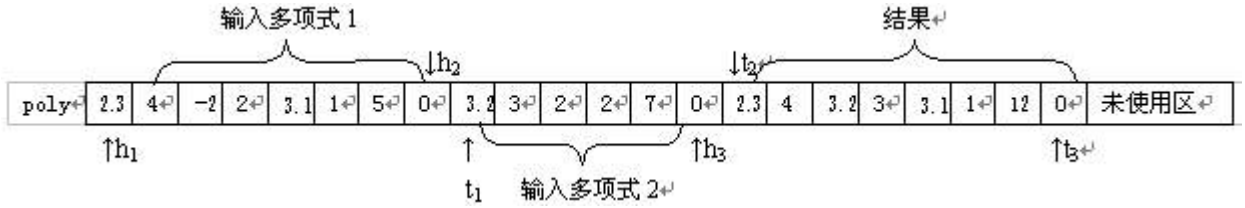
数组元素按下标随机访问,为了方便地实现访问操作,不论是一维还是多维数组,在计算机内总是按线性方式连续存放的。对二维数组通常有逐行存放与逐列存入两种实现方式。在本章中（如需要涉及存放方式）均假定是按行存放，即首先逐个存放第 1 行（在有些语言，如 C 中是第 0 行）的元素，再第 2，第 3 行等。换句话说，逐个看存储结构中的数组元素，下标值周期性变化，越右边的下标变得越快。对三维及三维以上的数组，情况也一样。

从这个意义上看，数组名实际上是指向该数组存储区首的指针，而下标表达式的值则表示指针增量。由于连续存放,对多维数组编译程序也很容易计算实际地址。在 C 语言训，数组名就被当作指针类型变理处理，因此数组名可以作函数的实在参数，并能在函数内改变作为实在参数的数组值。

[例 2．7]多项式加法。对两个最高次数可能很大，但非 0 系数不多的实系数一元多项式进行加法运算。

[解]一般来说，静态数组结构并不适合于实现多项式算法，应当采用后面章节中介绍的动态链表结构。不过对非 0 系数不多的多项式可以用静态数组结构与指针实现。
为了尽可能（静态结构局限下）有效地利用空间，采用一个数组 poly 存放两上输入多项式与一人输出多项式，用三对指针分别指出三个多项式首、末位置。

例如：（2.3x4-2x2+3.1x+5）+(3.2x3+2x2+7)=2.3x4+3.2x3+3.1x+12 其结构如下所示:



poly 中每个元素为一个记录,其值为多项式中一项的幂指数和系数.

多项式由用户从键盘输入,每次两个数分别表示多项式中一项的幂指数和系数.当输入(-1,0)时表示一多项式结束.为简单计,假定用户正确地按降幂输入.C 语言程序如下:

```
/ * 求两个多项式的和 */
#include <stdio.h>
#define MAXUM 100
```

```
struct term / * 多项式的一项 */
{
int exp;
float coef;
};
struct term poly [MAXUM],* h1 ,*h2 ,*h3 ,*t1 ,*t2 ,*t3 ;
void new_term();
main()
{
struct term *p,*q,*r,*I;
int m,minus;
flaot a,temp;
static char sym□={'','+'};
t1=h1=poly;/ * 输入第一个多项式 */
printf("Enter the first polynomial .\n term(expo coef )?\n");
scanf("%d%f".&m,&a);
while(m!=-1)
{
new_term(m,a,t1++);
printf ("term?\n");
scanf("%d%f",&m,&a);
}
t2=h2=t1;
printf("Enter the second polymonial ,\n term(expo coef)?\n");
scanf("%d%f".&m,&a);
while(m!=-1)
{
new_term(m,a,t2++);
printf ("term?\n");
scanf("%d%f",&m,&a);
}
t3=h3=t2;
p=h1;q=h2;r=h3;
while(p<t1)&&(q<t2))/ * 加法操作 * /
{
if(p->coef+q->coef)!=0)
new _term (p->exp,temp,r);
++p;++q;++r;
{
else if (p->exp<q->exp)/ * 幂次仅出现在一个多项式中,项送入结果 */
{
new_term (q->exp,q->coef,r);
++q;++r;
}
```

```
else
{
new_term(p->exp,p->coef,r);
++p;+=r;
}
}
while(p<t1)/ * 一个多项式项已处理完,另一个中余项送入结果 */
{
new-term(p->exp,p->coef,r);
++p;++r;
}
while(q<t2);
{
new_term(q->exp,q->coef,r);
++q;++r;
}
printf("\n");/ * 输出结果 */
printf("%c%6.2xˆ%d",sym[minus],I->coef,I->exp)
}
}
/ * 函数 new_term 送一个新项到多项式存区中 */
void new_term(m,a,pt)
int m; float a;
struct term * pt;
{
if(pt>poly +MAXUM)
{
printf("\n too many terms in polynomial.\n");
exit();
}
pt->coef=a;pt->exp=m;
}
```

注意，本例中函数 new_term 并没有建立新的存储结构，它只是生成项的一个新值送入已由数组 poly 说明例句创建的存储区中，这是静态结构。

2．2．2 矩阵运算

矩阵是重要的数学模型，广泛用于各种背景的计算问题。矩阵的一些常规运算如加法、乘法等，如按照运算基本定义，利用二维数组实现，方法非常直观。如何高效率实现大矩阵第乘法是很重要的问题，在许多算法设计教科书中有专节介绍，本节讨论其它一些比较特殊饶有趣味的矩阵问题。

1． 生成几种特殊矩阵

[例 2．8]生成 3-对角方阵。除主对角线及其上、下斜列以外其它元素均 0 的方阵称为 3-对角方阵，下面即为一例：

1 3 0 0 0 0
2 4 6 0 0 0
0 5 7 9 0 0
0 0 8 10 12 0
0 0 0 11 13 15
0 0 0 0 14 16

编一个程序首先生成如上所示的 6×6 的 3-对角阵，再以按行排列的方式将非 0 元素放入一个一维数组。

[解]n×n 3-对角阵中非 0 元素个数为 3n-2。除首、末行（列）外，每行（列）含三个非 0 元素。非 0 元素位置（I,j）满足条件|I-j|≤1。由此容易推知，按上图所示规律排列非零无素，元素值 aij 与所在位置下标值 I、j 关系为 aij=2j+I+1(I、j 计数由 0 开始)。

至于把 3-对角阵中非 0 元素传送到一维数组中的问题，只须注意到除首、末行外，每行中非 0 元素起始位置为 I-1，终止位置为 I+1，（I 为行号），则传送算法就很容易得到了.下面是用 C 语言编写的程序.

```
/ * 3-对角方阵的生成与一维存放 * /
# include <stdio.h>
# include <math.h>
#define SIZE 6
statin int a [SIZE][SIZE];
int b[3 * size-1];
main()
{
int,I,j,bi,count;
printf("\n 3-triagle matrix:\n");
for(I=0;I<SIZE-1;I++)
{
for(j=0;j<=SIZE-1;j++)
{
if(abs(I-j)<=1)
a[i][j]=2 * j+I+1
printf("%6d",a[i][j]);
}
printf("/n");
}
b[0]=a[0][0];
b[1]=a[0][1];
b[3 * SIZE-4]=a[SIZE-1][SIZE-2];
```

```
b[3 * SIZE-3]=a[SIZE-1][SIZE-2];
bi=2;
for(I=1;I<=SIZE-2;I++)
for(j=I-1;j<=I+1;j++)
{
b[bi]=a[i][j];bi++;
}
for(bi=0,count=0;bi<=3; * SIZE-3;bi++)
{
if((count++)==10)
{
printf("\n);count=0;
}
printf("%d",b[bi]);
}
}
```

在这个例子中,由于有简单的公式可以由下标 I、j 算出应放入该位置的值，因此按通常的行列号控制循环的方法是很方便。但是，若不易找出这样的公式，有时则以能过值来选择下标值比较容易实现。

[例 2．9]生成一个按蛇形方式排列自然数 1，2，……，N2 的 N 阶阵。

例如所要求的 7 阶方阵即为如下的方阵：

1	3	4	10	11	21	22
2	5	9	12	20	23	34
6	8	13	19	24	33	35
7	14	18	25	32	36	43
15	17	26	31	37	42	44
16	27	30	38	41	45	48
28	29	39	40	46	47	49

(a)

1	2	6	7	15	16	28
3	5	8	14	17	27	29
4	9	13	18	26	30	39
10	12	19	25	31	38	40
11	20	24	32	37	41	46
21	23	33	36	42	45	47
22	34	35	43	44	48	49

(b)

图 2.2 7阶方阵

[解]将自然数 1，2，……，N2 逐个顺序插入方阵中适当位置，这个过程沿斜列进行。将斜列编号为 0，1，2，……，2n（以 I 表记，n=N-1），从图 2.3 可见在一斜列上各元素的下标和是相等的,且等于斜列号 i.同方阵又可分上三角与下三角,上三角(含对角线)每一斜列上元素个数为 I+ 1 个;下三角每一斜列上元素个数为 2n-I+1 个.在斜列上安排数可以自右上向左下或自左下和右上两种方式时行,元素可表示成 a[I-j,j]或 a[j,I-j]形式,因此排数可由表 2.1 描述其方法.



图 2.3

表 2.1 蛇形方阵排数方法(1)

	左下向右上(↗)		右上向左下(↙)	
	下标变量	下标 j 的变化	下标变量	下标 j 的变化
上三角(含对角线)	$a[i-j][j]$	$0\to i$	$a[i-j][j]$	$i\to 0$
	$a[j][i-j]$	$i\to 0$	$a[j][i-j]$	$0\to i$
下三角	$a[i-j][j]$	$i-n\to n$	$a[i-j][j]$	$n\to i-n$
	$a[j][i-j]$	$n\to i-n$	$a[j][i-j]$	$i-n\to n$

对于图 2.2 蛇形方阵排数可走向可叙列号的奇偶性决定是↗或↙,对图 2.2(a)斜列号为奇数时为↗、而斜列号为偶数时为↙; 对图 2.2（b）则相反,即斜列号为奇数时为↙、而斜列号为偶数时为↗.

对应图 2.2(a)n=6(N=7),相应程序为:

```
#include<stdio.h>
#define size 10
int a[size][size];
main()
{int I,j,k,n;
prinf("\n input n\n");
scanf("%d",&n);
k=1;
for (I=0;I<=n,I++)
{if(I%2==1)
{for(j=0;j<=I;j++)
{a[I-j][j]=k;k++;
}
```

```
}
}
for(I=(n+1));I<=(2 * n);I++)
{if((I%2)==1)
{for(j=I-n;j<=n;j++)
{a[I-j][j]=k;k++;
}
}
else
{for (j=I-n;j<=n;j++)
{a[j][I-j]=k;k++;
}
}
}
printf("\n");
for(I=0;I<=n;I++)
{for(j=0;j<=n;j++)
printf("%3d",a[i][j]);
printf("\n");
}
}
```

蛇形方阵的又一种形式如图 2.4 所示.

28	29	39	40	46	47	49
16	27	30	38	41	45	48
15	17	26	31	37	42	44
7	14	18	25	32	36	43
6	8	13	19	24	33	35
2	5	9	12	20	23	34
1	3	4	10	11	21	22

(a)

22	34	35	43	44	48	49
21	23	33	36	42	45	47
11	20	24	32	37	41	46
10	12	19	25	31	38	40
4	9	13	18	26	30	39
3	5	8	14	17	27	29
1	2	6	7	15	16	28

(b)

图 2.4

从图 2.5 可见,在一斜列上各元素的下标差是相等的,且依次编为 6,5……,2,1,0,-1,-2,……-6(以 I 表记).每一斜列上元素个数为 n-I+1 个.称◣为下三角,◥为上三角,每一斜列上下标变量及下标表达式的变化见表 2.2.

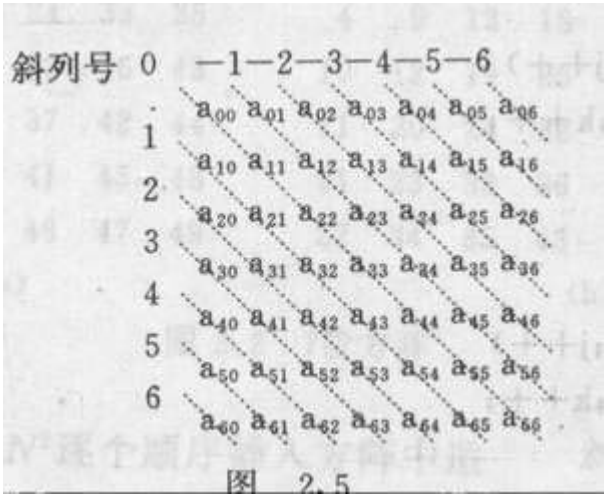


图 2.5

表 2-2 蛇形方阵排数方法(2)

	左上向右下(↘)		右下向左上(↖)	
	下标变量	下标 j 的变化	下标变量	下标 j 的变化
下三角（含对角线）	a[i+j][j]	0→n−i	a[i+j][j]	n−i→0
	a[j][j−i]	i→n	a[j][j−i]	n→i
上三角	a[j][j−i]	0→n+i	a[j][j−i]	n+i→0
	a[i+j][j]	|i|→n	a[i+j][j]	n→|i|

同样,对斜列上元素走向据斜列号 I 按 n+I 的奇偶性决定(用不着 n+I 而不用 I 可确保蛇形走向不会因 n 的奇偶性而变化).该题程序留作练习.

下面介绍蛇形方阵排数的又一种算法.

如图 2.2 和 2.4,排数从一个角走向另一对角时,每个后继元素是按斜列分四种情形时行的:↗与↙;或者↖与↘,这对应元素的下标变化分别是:

$$\begin{cases} i=i-1 \\ j=j+1 \end{cases} \quad \begin{cases} i=i+1 \\ j=j-1 \end{cases} \quad \begin{cases} i=i-1 \\ j=j-1 \end{cases} \quad \begin{cases} i=i+1 \\ j=j+1 \end{cases}$$

上述下标的变化可导致下标越界(I<0 或 I>n, j<0 或 j>n),这时就需调整 I 或 j,使之排数正常进行,下标变化规律见表 2.3.

表 2.3 蛇形方阵排数方法(2)下标变化规律

起止角	后继元素下标修正		对角的下标修正				边越界修正			
			转折	下标修正	转折	下标修正	$i<0$	$i>n$	$j<0$	$j>n$
$a_{00}\to a_{nn}$	↗	↙	$a_{nn}\to a_{n0}$	$\begin{cases}i=i+2\\j=j-1\end{cases}$	$a_{n0}\to a_{0n}$	$\begin{cases}i=i-1\\j=j+2\end{cases}$	$i=i+1$	$\begin{cases}i=i-1\\j=j+2\end{cases}$	$j=j+1$	$\begin{cases}i=i+2\\j=j-1\end{cases}$
$a_{n0}\to a_{0n}$	$\begin{cases}i=i-1\\j=j+1\end{cases}$	$\begin{cases}i=i+1\\j=j-1\end{cases}$	$a_{0n}\to a_{n-1,0}$	$\begin{cases}i=i+1\\j=j-2\end{cases}$	$a_{n0}\to a_{n-1,n}$	$\begin{cases}i=i-2\\j=j+1\end{cases}$	$\begin{cases}i=i+1\\j=j-2\end{cases}$	$i=i-1$	$\begin{cases}i=i-2\\j=j+1\end{cases}$	$j=j-1$
$a_{0n}\to a_{nn}$	↖	↘	$a_{nn}\to a_{0,n-1}$	$\begin{cases}i=i+1\\j=j+2\end{cases}$	$a_{0n}\to a_{n,n-1}$	$\begin{cases}i=i-2\\j=j-1\end{cases}$	$\begin{cases}i=i+1\\j=j+2\end{cases}$	$i=i-1$	$j=j+1$	$\begin{cases}i=i-2\\j=j-1\end{cases}$
$a_{n0}\to a_{n0}$	$\begin{cases}i=i-1\\j=j-1\end{cases}$	$\begin{cases}i=i+1\\j=j+1\end{cases}$	$a_{n0}\to a_{00}$	$\begin{cases}i=i+2\\j=j+1\end{cases}$	$a_{00}\to a_{n0}$	$\begin{cases}i=i-1\\j=j-2\end{cases}$	$i=i+1$	$\begin{cases}i=i-1\\j=j-2\end{cases}$	$\begin{cases}i=i+2\\j=j+1\end{cases}$	$j=j-1$

该算法程序也留作练习.

[例 2.10]生成一个螺旋方式排列自然数 1,2,……，N2 的 N 阶方阵。例如，7 阶螺旋方阵（n=N-1=6）：

```
 1  2  3  4  5  6  7
24 25 26 27 28 29  8
23 40 41 42 43 30  9
22 39 48 49 44 31 10
21 38 47 46 45 32 11
20 37 36 35 34 33 12
19 18 17 16 15 14 13
```

(a)

```
 1 24 23 22 21 20 19
 2 25 40 39 38 37 18
 3 26 41 48 47 36 17
 4 27 42 49 46 35 16
 5 28 43 44 45 34 15
 6 29 30 31 32 33 14
 7  8  9 10 11 12 13
```

(b)

[解]将自然数 1，2，……n,n2 逐个顺序排入方阵中适当位置，这过程从 a00 开始按图 2.7 所示外层到内层分别依上、右、下、左（图 2.6(a)）或左、下、右、上(图 2.6(b))排入.每进一层一行或一列的元素少 2 个,其变化规律见表 2.4(其中设层号 i(0→[(n-1)/2])

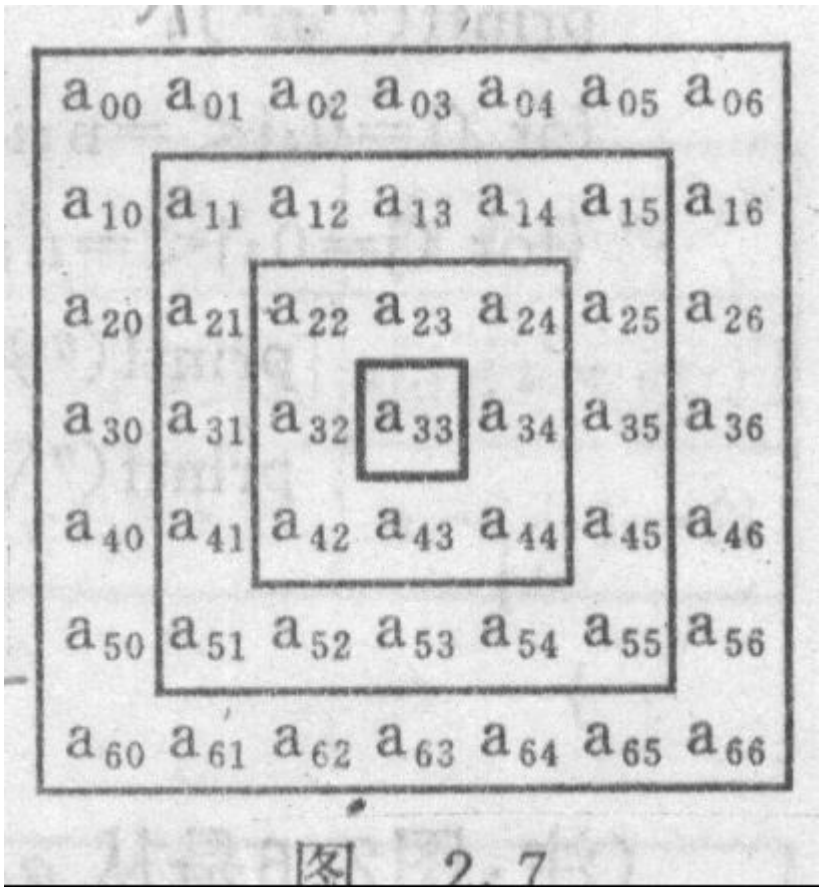


图 2.7

表 2.4 螺旋方阵排数方法

		上行	下行	左列	右列
顺时针	行	i	n-i	n-i→i+1	i→n-i-1
	列	i→n-i-1	n-i→i+1	i	n-i
逆时针	行	i	n-i	i→n-i-1	n-i→i+1
	列	n-i→i+1	i→n-i-1	i	n-i

对应图 2.6(a)(n=6),其程序为:

```
# include<stdio.h>
# define size 10
int a [size][size];
main()
{int I,j,k,n;
printf("\n input n\n");
scanf("%d",&n);
```

```
k=1;
for(I=0;I<=(n-I-1)/2;I++)
{a[j][n-i]=k;k++;
}
for(j=n-I;j>=(I+1);j--)
{a[n-I][j]=k,k++;
}
for(j=n-I;j>=(i+1);j--)
{a[j][i]=k;k++;
}
}
if(n%2==0)
{a[n/2][n/2]=k;
}
printf("\n");
for(I=o;I<=n;I++)
{for(j=0;j<=n;j++)
printf("%3d",a[i][j]);
printf("\n");
}
}
```

(注:图 2.6 所示是从 a00 开始旋入,也可从另三个角旋入,或者从内向外旋出).

[例 2.11]生成一个双螺旋方式排列自然数 1,2,……, N2 的 N 阶方阵。例如，7 阶双螺旋方阵（n=N-1=6）.

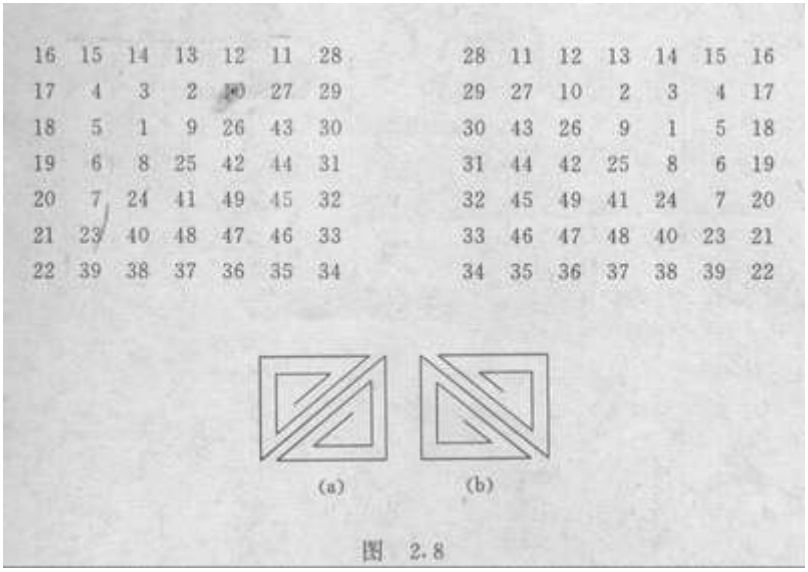


图 2.8

[解]该例图 2.8(a)左上三角为逆时针旋入.仍如[例 2.10]一样,按层次进行考虑,上、下三角
都可看成有[(n+1)/3] 层。在分析多个 n 阶方阵后可知对图 2.8(a)一类的方阵

上三角旋出的层次为[(n+1)/3],在 n 对 3 的模为 1 时有一个孤立元素(两个下标均为[n/3]),每层中按↗、←、↓的次序排数，每串上有 n-3I+2 个元素（I 为层号）；下三角旋入层次为[n/3]层 ，每层中按↓、←、↗的次序排数，每串上也有 n-3I+2 个元素，在下三角中当 n 对 3 模为 1 时也有一个孤立元素（两个下标均为 n-[n/3]），其变化关系见表 2.5。对图 2.8（b）一类的方阵上三角旋出的层次为 [(n+1)/3]→1,在 n 对 3 的模式为 1 时有一个孤立的元素(下标为([n/3],n-[n/3]))，每层中按↖、→、↓的次序排数，每串上有 n-3I+2 个元素；下三角旋入层次为 1→[(n+1)/3] ,每层中按↓、→、↖次序排数,每串上也有 n-3I+2 个元素,在下三角旋入中当 n 对 3 的模为 1 时也有一个孤立元素(下标(n-[n/3],[n/3]))，其变化关系见表 2.6.

表 2.5 对应图2.8(a)的关系表

	左上三角↗	左上三角←	左上三角↓	右下三角↓	右下三角←	右下三角↗
下标变量	a[n-i-j][j]	a[j-1][j]	a[j][j-1]	a[j][n-i+1]	a[n-i+1][j]	a[j][n+i-j]
j的变化	i→n-2i+1	n-2i→i-1	i→n-2i+1	2i-1→n-i	n-i+1→2i	n-i+1→2i

表 2.6 对应图2.8(b)的关系表

	右上三角↖	右上三角→	右上三角↓	左下三角↓	左下三角→	左下三角↖
下标变量	a[i-j][j]	a[i-1][j]	a[j][n-i+1]	a[j][i-1]	a[n-i+1][j]	a[i+j][j]
j的变化	n-i→2i-1	2i→n-i+1	i→n-2i+1	2*i-1→n-i	i-1→n-2i	n-2i+1→i

```
#include<stdio.h>
#define size 10
int a[size][size];
main()
{int,j,j,k,n;
printf("\n input n\n");
scanf("%d",&n);
k=1;
if(n%3==1)
{a[n/3][n/3]=k;k++;
}
for(I+(n+1)/3;I>=1;I--)
{for (j=I;j<=(n-2*I+1);j++)
{a[n-I-j][j]=k;k++;
}
for(j=n-2*I;j>=(I-1);j--)
{a[I-1][j]=k;k++
{
for(j=I;j<=(n-2*I+1;j++)
{a[j][j-1]=k;k++
}
}
```

```
for(I=0;I<=n;I++)
{a[n-i][i]=k;k++;
}
for(I=1;I<=((n+1)/3;I++)
{for(j=2*I-1;j<=(n-i);j++)
{a[j][n-I+1]=k;k++;
}
for(j=n-I+1;j>=(2*i);j--)
{a[j][n+I-j]=k;k++;
}
}
if(n%3==1)
{a[n-n/3][n-n/3]=k;
}
printf("\n");
for(I=0;I<=n;I++)
{for(j=0;j<=n;j++)
printf("%3d",a[i][j]);
printf("\n");
}
}
```

2. 矩阵旋转

矩阵的转置与旋转都是常见的运算.转置是使一个方阵以主角线为轴翻转.运算前后两个矩阵(设为 A 与 A′)元素满足 aij=a′ ji,因此用数组实现很简单.而矩阵旋转指使整个方阵顺时针或逆时针转 90°,如图 2.9 所示.

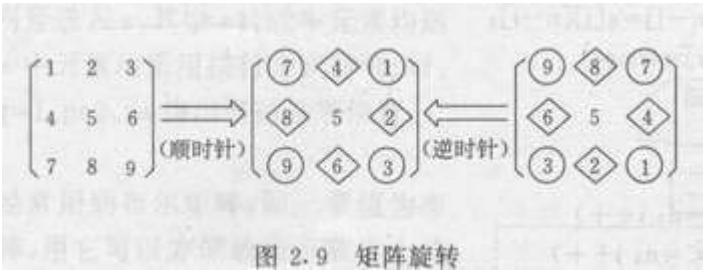


图 2.9 矩阵旋转

[例 2.12]将方阵逆时针旋转 90°.

[解]矩阵旋转可将矩阵分层进行,分层如图 2.10 所示,对 n+1 阶矩阵可分成 [(n+1/2)] 层.用 I 表记
(其值 0→[(n-1)/2]).每层中将元素分成 n-2I 组,每组中 4 个元素,例如图 2.10 中 I 表记为 1 层(即从外向内数的第 2 层),其中含 n-2·I=4 组(a11、a15、a55、a51)、(a12、a25、a54、a41)、(a13、a35、a53、a31、)、(a14、a45、a52、a21)旋转便是对每组 4 个元素循环移动一次(顺时钟为右移、逆时针为左移)。分析每组元素,设任一组为(ai1j1、ai2j2、ai3j3、ai4j4)其中有 如下关系,j1=i2、j2=i3、j4=i1;I1+i3=j1+j3+i2+i4=j2+j4=n.。

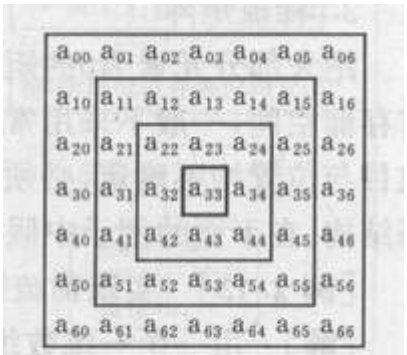


图 2.10

根据上述规律，矩阵顺时针旋转程序为：

```
#include<stdio.h>
#define size 10
int a [size][size];
main()
{int I,j,k,n,temp;
printf("\n input n\n");
scanf("%d",&n);
for(I=0;j<=;I++)
{for(j=0;j<=n;j++)
{a[i][j]=I*(n+1)+j;
}
}
printf("\n");
for(I=0;I<=n;I++)
{for(j=0;j<=n;j++)
pritf("%3d",a[i][j]);
printf("\n");
}
for(I=0;I<=(n-1)/2;I++)
{for(j=I;j<=(n-I-1);j++)
{temp=a[i][j];
a[i][j]=a[n-j][i];
a[n-j][i]=a[n-i][n-j];
a[n-i][n-j]=a[j][n-i];
a[j][n-i]=temp;
}
}
printf("\n");
for(I=0;I<=n;I++)
{for(j=0;j<=n;j++)
```

```
printf("%3d",a[i][j]);
printf("\n");
}
}
```

3.稀疏矩阵

绝大部分元素为 0 的矩阵称为稀疏矩阵.应用中往往使用阶数最高的稀疏矩阵,为了节省存储窖,一般不采用常规的数组按行列存放方式,而仅仅保存非 0 元素.因为非 0 元素的数目与位置均不确定,必须同时保存非 0 元素的位置信息及值,而且多数有效的方法采用动态结构.在下面的例子中假定非 0 元素个数不超过一个定值,用静态数组来实现稀疏矩阵.

[例 2.13]实现稀疏矩阵加法,见图 2.11.

[解]用一个一维数组存放稀疏矩阵,其元素为记录:

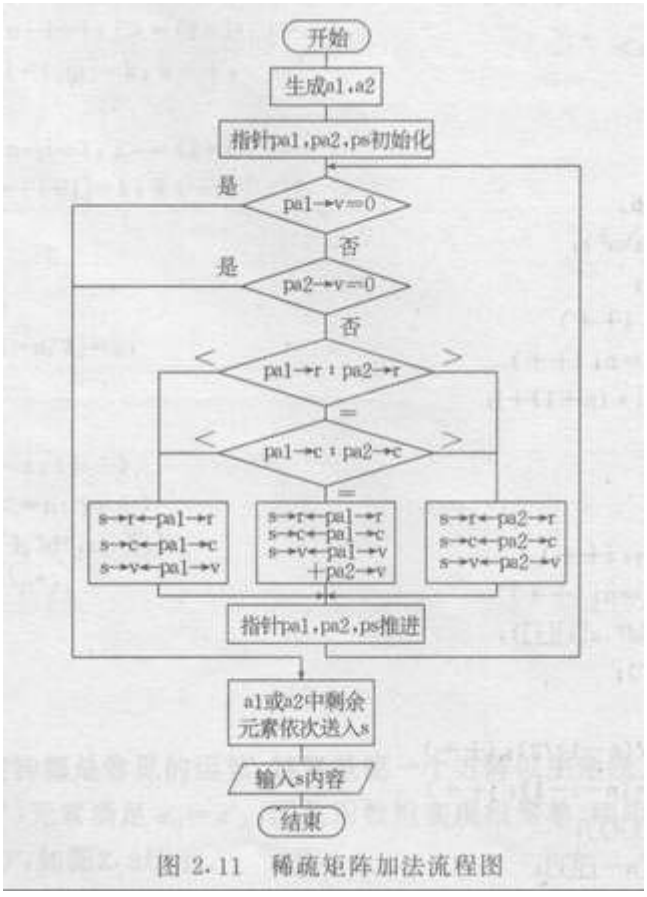


图 2.11 稀疏矩阵加法流程图

输入两个稀疏矩阵 a1、a2。加法运算结果为 s. 当 a1、a2 中出现相同的行、列呈时，相应的两个值相加，连同行、列号送入 s ，其余 a1、a2 中元素均送入 s。要求 a1、a2、s 各有一指针 pa1、pa2、ps 指出当前处理位置。

4. 布尔矩阵

程序设计中经常用到布尔矩阵，即元素值为布尔值（0 或 1）的矩阵，用它可以方便地表示数学上的关系模型。

[例 2．14]判定给定关系的性质。

[解]考虑含 16 个元素的集合 S ={a0,a1,……,a15}上的关系 R,R 用 16×16 布尔矩阵表示，则 R 可实现为二维字位数组，即一维 16 位无符号数数组。集合元素 ai、aj；满足关系 R ,当且仅当（I,j）位置上数组元素为 1

假定只要求判断 R 是否为等价关系，则需判断下列性质是否满足：

（1） 自反性：即关系矩阵主对角元素均为 1；

（2） 对称性：即关系矩阵与转置矩阵应当相等；

（3）传递性:关系矩阵与其传递闭包矩阵应当相等。R 的传递闭包 $R^+=R+R^2+\cdots\cdots R^n$（n 为 R 的阶）。

对每一性质编写一个判别函数。其中判别对称性主要完成求转置矩阵的操作，以字位数组实现矩阵转置算法流程见图 2。12。

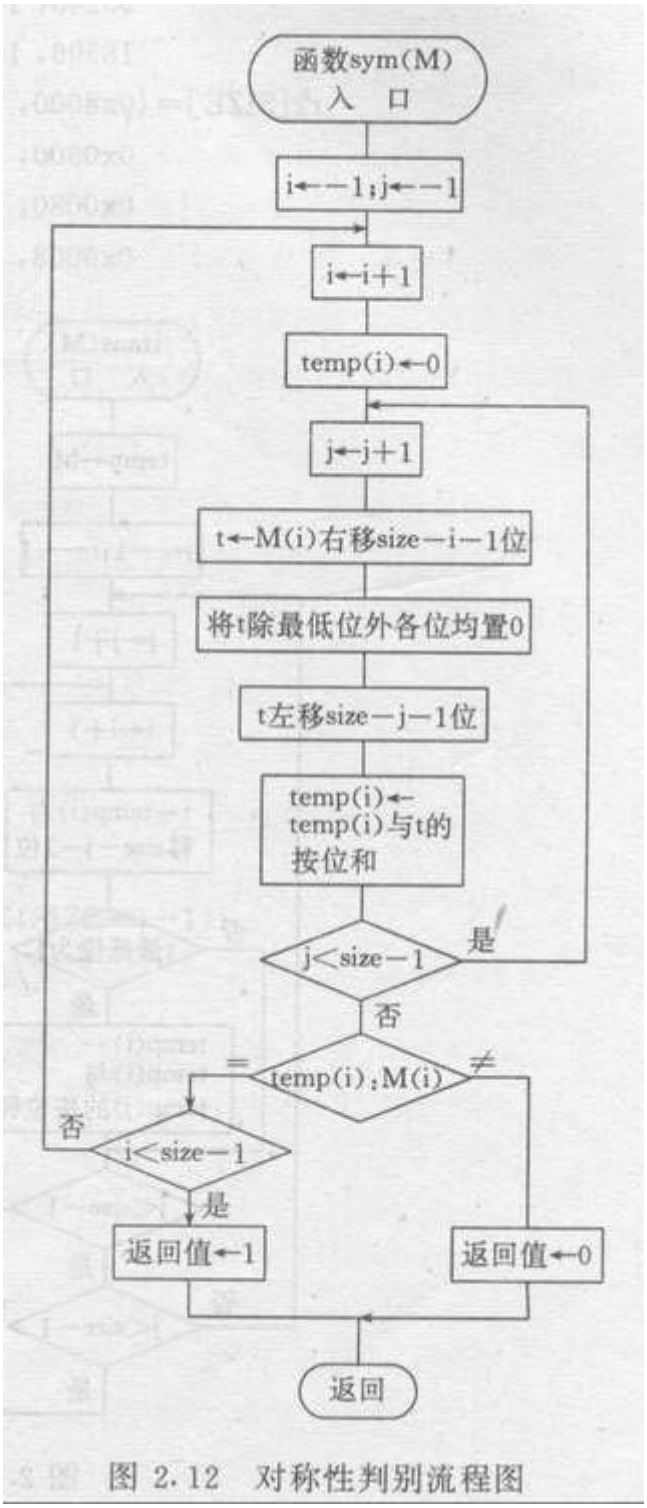


图 2.12 对称性判别流程图

求传递闭包采用 Washall 算法，它利用布尔运算的特征，避免了通常矩阵乘法所需的大量计算。注意，根据上述传递闭包 R^+计算公式，R^+可以看作是在 R 中增加非 0 项的过程，而且一旦某项成为非 0，最终结果中该项必保持非 0，逐列检查 R，在第 j 列上，如发现第 I 个元素（rij）为 1，则表示 ai 与 ai 满足关系 R^+，此时如果在 j 行上有某个元素 rik=1 即 aj 与 ak 满足关系 R ，由定义 ai 与 ak

必满足关系 R^+ .因此在结果矩阵中（I,j）位置上元素应为 1，可以用把 j 行按位加到 I 行上的操作实现这一点。传递性判别流程图见图 2。13。

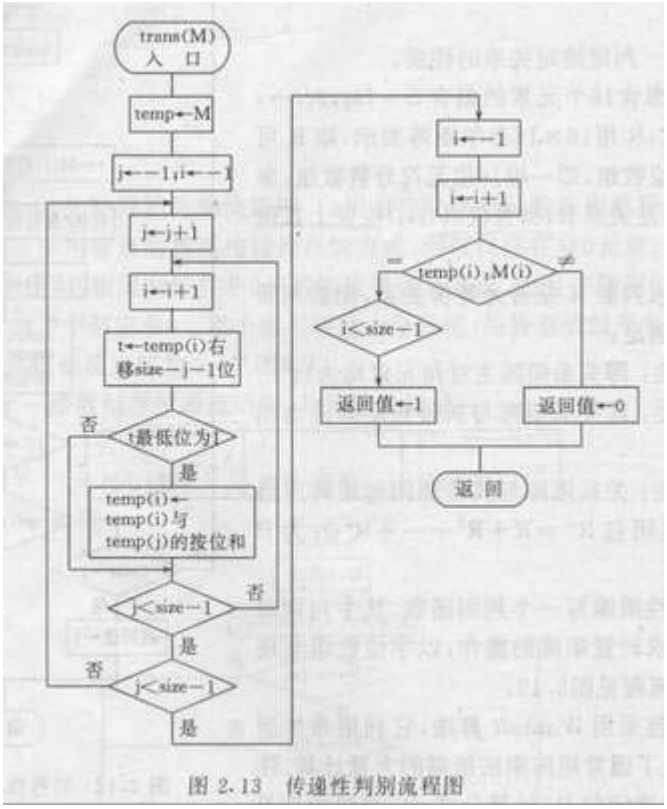


图 2.13 传递性判别流程图

以下是 C 语言程序.

```
/ * 判别等价关系 */
#include<stdio.h>
#define SIZE 16
#define r r2 / * 选择调试数据 */
static unsigned r1[SIZE]={35232,17160,8257,4098,/ * 数据 1* /
34888，1284，19475，52864，
33248，10456，32976，112，
18506，1028，4106，8705}，
r2[SIZE]={0x800,0x4000,0x2000,0x1000,/ * 数据 2 * /
0x0800,0x0400,0x0200,0x0100,
0x0080,0x0040,0x0020,0x0010
0x0008,0x0004,0x0002,0x0001};


Main()
{
if(ref(r)&&sym(r)&&trans(r))
printf("\n Yes,R is an equivalene relation.\n");
```

```
else
printf("\nSorry? R is not an equivalence relation.\n");
}
/ * 函数 ref 判别自反性 * /
int ref(m)
unsigned m[SIZE];
{
int I;
for(I=SIZE-1;I>=0;I--)
{
if ((m[i]>>I)&1==0)
return(0);
}
printf("\nreflexible\n");
return(1);
}
/ * 函数 sym 判别对称性 * /
int sym(m)
unsigned m[SIZE];
{
int I,j;
for(I=0;I<SIZE;I++)
temp[i]=0;
for(I=0;I<SIZE;I++)
{
for(j=0;J<SIZE;I++)
{
for(j=0;J<SIZE;j++)
temp[i]=((m[j]>>(SIZE-I-1)&1)<<(SIZE-j-1);
if(m[i]!=temp[i])
return(0);
}
printf("\n symetricn");
return(1);
}
/ * 函数 trans 判别传递性 * /
int trans(m)
unsignedm [size];
{
unsigned temp[size];
int,I,j;
for(I=0;I<size;i++)
temp[i]=m[i];
for(j=0;j<SIZE;j++)
```

```
for(I=0;I<SIZE;I++)
if(((temp[i]>>(SIZE-1-j))&1)==1)
temp[i]=temp[j];
for(I=0;I<SIZE;I++)
if(temp[i]!=m[i])
return(0);
return(1);
}
```

2.2.3 趣味程序设计问题

利用数组结构可以解许多趣味问题，这些问题虽然表面上没有什么实用意义，却是良好的训练材料，常出现在各种教科书中。

[例 2. 15]生成魔方阵。

[解]魔方阵是指元素为自然数 1，2……，N2 的 N×N 方阵，每个元素值均不相等，每行、每列以及主、副对角线上各 N 个元素之和却相等。例如图 2。14 即为 3×3 魔方阵。

对奇阶魔方阵，可用 Dole Rod 算法生成。其过程为：从 1 开始，依次插入各自然数，直到 N2 为止。选择插入位置原则如下：（参见图 2。14）

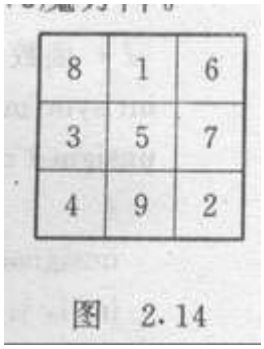


图 2.14

·第一个位置在第一行正中。

·新位置应当处于最近一个插入位置右上方，但如右上方位置已超出方阵上边界，则新位置取应选列的最下一个位置；如超出右边界则新位置取应选行的最左一个位置。

·若最近一个插入元素为 N 的整数倍，则选下面一行同列上的位置为新位置。

用二维整型数组实现方阵，C 语言程序如下：

```
/ * 生成奇数阶魔方阵 * /
#include<stdio.h>
```

```
#define MAXSIZE 15 / *最大允许阶数 * /
int magic[MAXSIZE][MAXSIZE];
int cur_I=0,cur_j;
main()
{
int count,size=0,I,j;
while((size%2)==0) / * 输入阶数，只接受奇数 * /
{
printf("\nenter ssquare size(ODD number):");
scanf("%d",&size);
}
cur_j=(size-1)/2;
for(count=1;count<=size * size;count++)
{
magic[cur_i][cur_j]=count;
if((count%size)==0)
{
cur_I=cur_I-41;
cur_j=cur_j+1
if(cur_I<0)
cur_I+=size;
else if ((cur_j==size)
cur_j-=size;
}
for(I=0;I<size;I++)
{
printf("\n");
for(j=0;j<size;j++)
printf("%3d",magic[i][j]);
}
}
```

奇数阶魔方阵的这一算法第一位置还可在另外三边之正中，走向也可右上、右下、左上、左下地变化，此处右上是 I 为-1，j 为+1，而右边中间开始右下就为 I 与 j 都是+1。

[例 2．16]幸运方阵问题。

所谓"幸运方阵"问题是这样的，任意指定一个阶数，例如 3；再任意选定一个"幸运数"，例如 100，要求生成一个 3 阶方阵。从方阵中任意划去一行与一列，记下交叉点的数值；再从方阵剩余部分任意划去一行一列，再记下交叉点数值；继续这一过程，当方阵已不剩任何元素时，所有记下的元素值之和恰好为 100。下面的方阵即为所需结果（结果不唯一）。

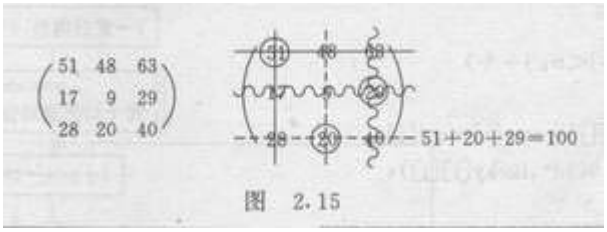


图 2.15

[解]表面上看，方阵元素之间似乎没有什么规律性联系。但实际上，按规定的划法，所有记下的交叉点元素两两均异行异列，而且这些元素的全体恰好涉及全部行与列。如果每行每列均给一个隐含的数值"名",选用 6 个名时,要求 6 个数值之和恰好为 100;而每个位置上真的元素值选其行、列"名"之和，显然生成的方阵满足条件。注意，实际上给定一个阶数 n 与一个幸运数 luckly，可以有无数多上解。如果不希望多出现负数，lucky 相对于 n 不应过小。

在下面的程序中用二维整型数组表示方阵，两个一维数组 row、col 存行、列"名"，为了使方阵元素看上去无规律，方阵中除最后一个列名外其余行、列名均为随机数，最后一个列名由减法得到。不同的随机数范围会影响结果中负数的出现情况。下面即为 C 语言程序：

```
/ * 生成幸运方阵 */
#include <stdio.h>
#include <math.h>
#define SIZE 10
main()
int luckly[SIZE][SIZE],row[SIZE],
col[SIZE],n,luckly_n,k,sum,I,j;
printf("\n enter the size of LUCKY square\n");
scanf("%d",&n);
printf("enter your LUCKY number\n");
scanf("%d",*lucky_n);
srand(rand()%5);
k=lucky_n/n;
if(k==0)
printf("\n your LUCKY number too small.\n");
sum=0;
for(I=0;I<n;I++)
{
row[i]=rand()%k;
col[i]=rand()%k;
sum=sum+row[i]+col[i];
}
col[n-1]=col[n-1]-(sum-lucky_n);
printf("\n");
for(I=0;I<n;I++)
```

```
{
for(j=0;j<n;j++)
{
lucky[i][j]=row[i]+col[j];
printf("%5d",lucky[i][j]);
}
printf("\n");
}
}
```

国际象棋的棋盘是由 8×8 个方格组成的方阵，许多趣味问题与棋盘有关，例如八皇后问题就是一个著名的棋盘布子问题。这里讨论一个与走马有关的问题。

马是国际象棋中走法比较特别的子，走法如图 2.16 所示,在不受棋盘边界限制时,◎处的马能且只能走到标△的方格之上.

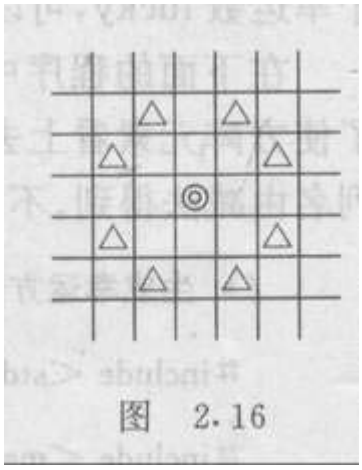


图 2.16

[例 2.17]马的遍历问题.从任意指定的方格开始,寻找一条路径,走遍全棋盘且每格只经过一次(不必回出发点).程序输出 8×8 方阵,用 1～64 表示走过每一格子的次序,起点标为 1.

[解]用二维整型数组 board 表示棋盘,其元素值记录经过相应位置的步序号.当前位置的行、 列
号用 cur_row、cur_col 表示。

按照马的走法，如当前位置为（I,j)，下一个可能位置必为（I+2,j±1）、（I+1，j±2）、(I-2,j ±1)八个位置中的全部或部分。实际可能位置是指按步法可能走到的棋盘范围内尚未经过的位置（值为 0），又称为出口。

为便于循环控制,用两个一维数组存放八个可能位置对于当前位置的行、列增量，注意两个数组元素必须互相对应：

```
move1={2,2,1,1,-1,-1,-2,-2}
```

move1={1,-1,2,-2,2,-2,1,-1}

采用 Warnsdoff 策略求解，即每走一步都选择出口最少的位置（当然必须是允许走到的位置）。比方说，当前位置为 A 格，A 有三个出口 B、C、D，它们各自的出口数为 3、2、4，则应选择 C 为下一个当前位置。

函数 nexttpos 检查指定位置的出口情况，返回该位置出口个数，并依次将各出口位置的行、列号填入数组 nexti 与 nextj。

当出口数为 1 时直接推进；若出口数大于 1，调用函数 movenext 选择新的当前位置。Movenext 当前位置的诸出口（已记录在 nexti 与 nextj 中）逐个调用函数 nextpos，始终记录已知出口数最少的格子位置及其出口的信息，最终值即为新的当前位置的相应信息。Movennext 返回新当胶位置出口个数。

图 2。17-图 2。19 描述了主要算法流程。

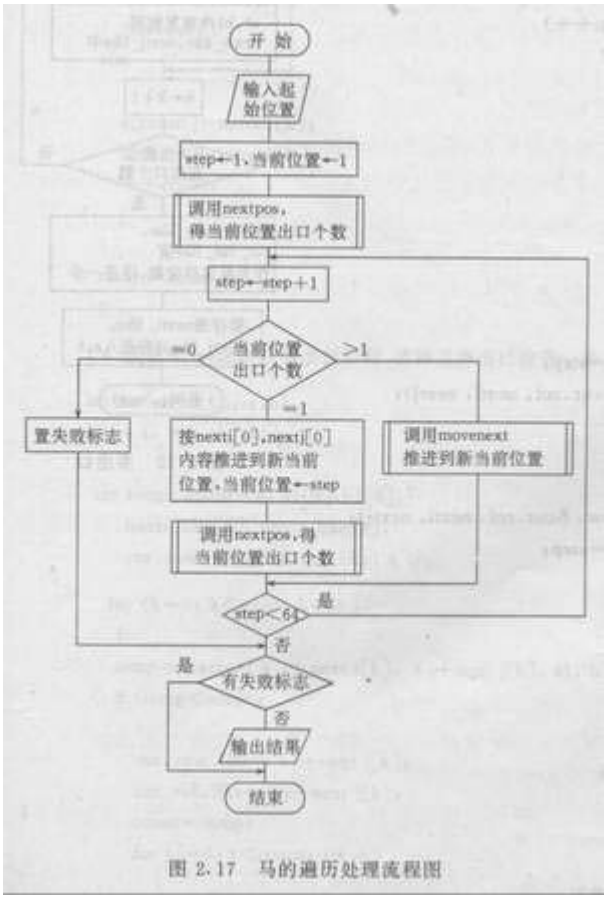


图 2.17 马的遍历处理流程图

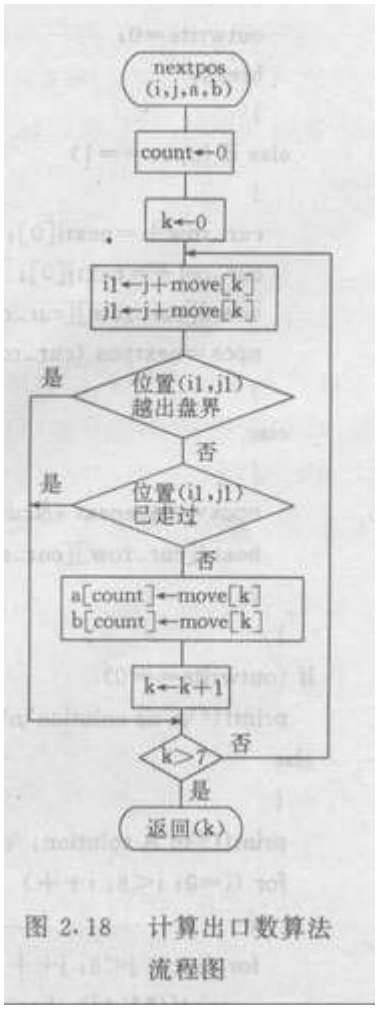


图 2.18 计算出口数算法流程图

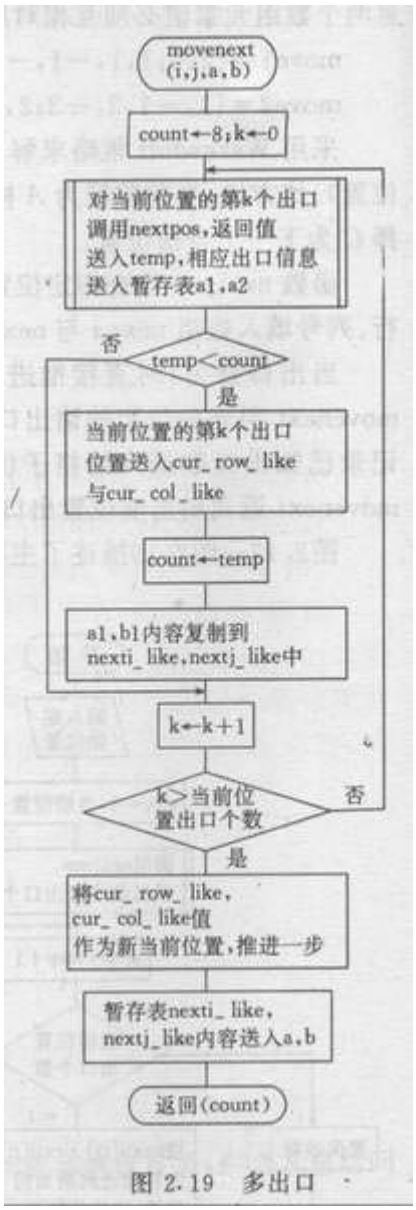


图 2.19 多出口

```
/ * 国际象棋盘上马的遍历 * /
#include<stdio.h>
static int movel[8]={2,2,1,1,-1,-1,-2,-2,},
move2[8]={1,-1,2,-2,2,-2,1,-1},
board[8][8]={{0}};
int nexti[8],nextj[8],npos,cur_row,cur_col,step,outwrite=1
main()
{
int,I,j;
printf("\nEnter the starting position(row col):");
scanf("%d%d",&cur_row,&cur_col);/ *输入起始位置 * /
board[cur_row][cur_col]=1;
npos=nextpos(cur_rowm,cur_col,nexti,nexti);
```

```
for(step=2;step<=64;step++)
{
if (npos==0)
{
outwrite=0;
break;
}
else if(npos==1)
{
cur_row+=nexti[0];
cur_col+=nextj[0];
board[cur_row][cur_col]=step;
npos=nextpos(cur_row,cur_col,nexti,nextj);
}
else
{
npos=movenext(&cur_row,&cur_col,nexti,nextj);
board[cur_row][cur_col]=step;
}
}
if(outwrite==0)
printf("\n no solution\n");
else
{
printf("\n A solution:\n");
for(I=0;I<8;I++)
{
for(j=0;j<8;j++)
printf("%4d",board[i][j]);
printf("\n");
}
}
}
/ * 函数 nextpos 计算位置（I,j）出口个数 * /
int nextpos(I,j,a,b)
int I,j,a[8],b[8];
{
int count=0,k,i1,j1;
for(k=0;k<8;k++)
{
i1=I+move[k];
j1=j+move2[k];
if(i1>7||i1<0||j1>7||j1<0)
continute;
```

```
else
{
a[count]=move1[k];
b[count]=move2[k];
++countp;
}
}
return(count);
}
/ *函数 movenext 在有多出口时，选择适当的出口推进一步 */
int movenext(I,j,a,b)
int * I, * j,a[],b[];
{
int temp,count=8,a1[8],b1[8],
nexti_like[8],nextj_like[8],
cur_row_like,cur_col_like,k,t;
for(k=0;k<npos;k++)
{
temp=nextpos( * I+nexti[k],* j+nextj[k],a1,b1);
if(temp<count)
{
cur_row_like=* I+nexti[k];
cur_row_like=* j+nextj[k];
count+temp;
for(t=o;t<count;t++)
{
next I_like[t]=a1[t];
next j_like[t]=b1[t];
}
}
}
* I=cur_row_like;
* j=cur_col_like;
for(k=0;k<count;k++)
{
a[k]=nexti_like[k];
b[k]=nextj_like[k];
}
return count;
}
```

注意，最少出口数的可选位置不一定唯一。为简单计，上面程序中选第一个遇到的（实际上由两上 move 数组中元素给定次序确定），这个算法不保证一定能找到解，不过找不到的可能性很小。

在本例中很好地利用了数组结构的特点：

·走马需随机地访问棋盘上不连续的位置，通过数组下标变化很容易实现。

·把马移动规则组织为位移量数组，使得在任何位置上马的各出口能采用循环语句处理，这是很有用的技巧。

2.3 队列

2．3．1 队列的特征与基本操作

队列是日常排队现象抽象出来的一种数学模型。队列结构中的数据项数可以随机地增减，但数据进出队列的方式是受限制的：数据只能从队列尾部插入，也只能从首部取出，已进入队列的数据次序不能再改变。每当有数据被取出，剩下的便依次向首部移动，因此队列被称为是一种"先进先出"（FIFO）结构。在任何时刻队列中唯一能被读到的数据是首端数据。

根据上述特征，队列有如一基本操作：

·创建（CreateQ）.

·初始发（InitQ）：按约定置队列为空状态。

·数据插入（EnQ）：队列尾端加入一个新数据项。

·数据取出（DeQ）：从队列首部取出一个数据项，并使余下诸项向队列首部移动。

·队列空（EmptyQ）：判别队列是否为空。
从概念上说，队列不存在"满"状态，其长度可以任意增加，但实现（不论静态或动态）中总有窖限制，因此有：

·队列满（FullQ）：判别队列是否已满。

2．3．2 队列的实现

队列与后面将讨论的栈都是程序设计中最常用的结构，但与记录、数组等不同，绝大多数语言中并未提供直接实现这些结构的设施。

要实现队列长度任意变化的特征，使用动态结构比较合适，本节中只讨论用静态数组类型实现队列的一种方法。

[例 2．18]用数组实现队列结构。

[解]假设队列中元素类型为 T。估计队列最大长度为 maxize，在任一时刻，队列首、尾位置分别用指针 head、tail 指出，则队列可以用下面的记录结构实现：



队列初始状态为：head=0;tail=-1.

根据队列定义，head 值应恒为 0，那么每当队列中送出一个数据项，必须执行多次移动操作。

采用"循环数组"技术可以完全避免队列内元素的移动,即当插入一个数据项时先移动 tail(即 tail=(tail+1)mod maxsize),插入的数据项存 qdata[tail]。从队列中取数据项也是先移动 head(即 head=(head+1)mod maxsize),qdata[head]就是取出的数据。（注意：head 和 tail 的移动都是加 1 后模 maxsize .）在移动中，若 head(或 tail)值为 maxsize-1。则移动后 head(或 tail)的值变成 0 了。对这种特征可以想象为 Qdata 存区尾单元与首单元"粘接"起来，形成一个环状（如图 2。20 所示），指针值按一个指定方向可以无终止地变化。（注：对 x=(x+1)mod n 则 x 在[0，n-1]中变，而 x=x mod n+1,则 x 在[1，n]中变。）

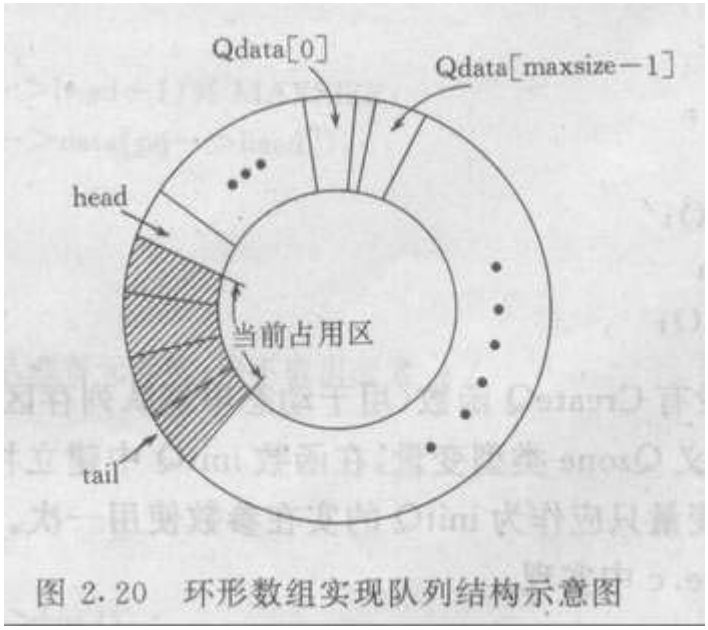


图 2.20 环形数组实现队列结构示意图

用"循环数组"实现队列，必须注意怎样判别队列空与满状态，在以下的实现中，除起始状态外，任何时刻 tail 所指为最近一个进入队列的元素，而 head 所指的是最近一个已经离开队列的元素原先所占的位置（见图 2。7）。因此（head+1）mod maxsize 才是真正当前队列中首元素位置。采用条件：

（tail+1）mod maxsize=head

作为"队列满"条件，实际上此时队列中还有一个空位置，这样队列存区有效空间比定义的最大空间少一个单元。假如把这个单元也利用上，则当 head 与 tail 指向同一单元时，既可能是"满"也可能是"空"，必须根据 tail"追上"了 head，还是 head 追上 tail 才能区分，这要给处理带来不便。

在风格良好的程序中对抽象数据类型的操作只能通过规定的子程序组执行，这些子程序除实现基本操作外也可适当增加其它需要的操作，例如：

·查看队列头（ExamQ）：读队列首项的值但无数据队列。

下面给出程序。它首先将各种类型定义和函数说明放在文件 queue.h 中，这里 T 暂定为 int，应用时按需要改第一个 typedef 语句。对队列的操作通过一个指向 Qzone 类型的指针进行，该指针类型即为 Q .

```
/ * 抽象数据类型队列定义模块 */
/ *queue.h * /
#define MAXSIZE 100
typedef int T;
typedef int Qdata[MAXSIZE];
typedef struct
{
Qdata data;
Int head,tail;
}Qzone;
typedef Qzone * Q;
typedef enum{true=1,false=0}bool;
typedef enum{success,fail}status;

externQ Q initQ ();
extern status enQ();
extern T * deQ ();
extern T * examQ ();
extern bool fullQ ();
extern bool emptyQ ();
```

由于是静态实现，没有 CreateQ 函数（用于动态申请队列存区并建立相应的指针值）。在应用模块中必须显示定义 Qzone 类型变量，在函数 initQ 中建立相应的指针值。在应用模块中，每一个 Qzone 类型变量只应作为 initQ 的实在参数使用一次。

各操作在模块 queue.c 中实现。

```
/ *队列操作实现模块 * /
/ * queue.c * /
#include "queue.h"
/ *函数 initQ 建立一个队列指针值*/
QinitQ (a)
Qzone * a;
{
QP;

P=a;
p->head=MAXSIZE-1;
p->tail=MAXSIZE-1;
return(p);
}
/ * 函数 enQ 将元素 term 加入队列中 * /
status enQ(term,pq)
T * term ;Q pq;
{
if(((pq->tail+1)%MAXSIZE)==pq->head)
return fail;
else
{
pq->tail=(pq->tail+1)%MAXSIZE;
pq->data[pq->tail]= * term;
return success;
}
}
/ * 函数 deQ 从队列中取出一元素 * /
T * deQ(pq)
Q pq;
{
T * temp;
if(pq->tail!=pq->head)
{
pq->head=(pq->head+1)%MAXSIZE;
temp=&(pq->data[pq->head]);
return temp;
}
}
/ * 函数 examQ 读出队列首素值，但不取出元素 * /
T * examQ (pq)
Q pq;
{
T * temp;
```

```
If (pq->tail!=pq->head)
{
temp=& (pq->dqta[pq->head+1%MAXSIZE]);
return temp;
}
}
/*函数 fullQ 判别队列满状态*/
bool fullQ(pq)
Q pq ;
{
return (((pq->tail+1)%MASIZE)= =pq->head);
}
/*函数 emptyQ 判别队列空状态*/
bool emptyQ (pq)
Q pq ;
{
return (pq->tqil = =pq->head);
}
```

应当指出，若队列空，则函数 DeQ 与 ExamQ 的近回值均无意义。应用模块在调用这两个函数前应当先查看队列是否空（用 EmptyQ）.

2．3．3 队列的应用

队列的结构经常用于计算机模拟分析现实生活中的排队现象，这是计算机辅助决策的重要内容之一。先看一个简单的例子。

[例 2。19]模拟服务台前的排队现象。

[解] 作为简单模拟，假定只有一个服务窗口，只排一个队，而且每个顾客所需实际服务时间为定值。顾客到达时间是随机的。

程序输入为：顾客密度概率（每一单位时间到一个顾客的概率。所谓单位时间是相以的，概率值必须小于 1）；每位顾客所需实际服务时间；模拟时间段总长（按单位时间计）。

程序输出为：总服务人数以及每个顾客平均排队等待时间（不包括实际服务时间）。

用队列 wait 表示等待队列，顾客用整数表示，其值为该顾客进入队列时间（显然当窗口空闲时，进队列，进队列时间等于出队列时间）。时钟用一整型数 time 表示，其初值为 0。在本例中时钟每步长以应实际时间 1 分钟。

算法流程见图 2。21。

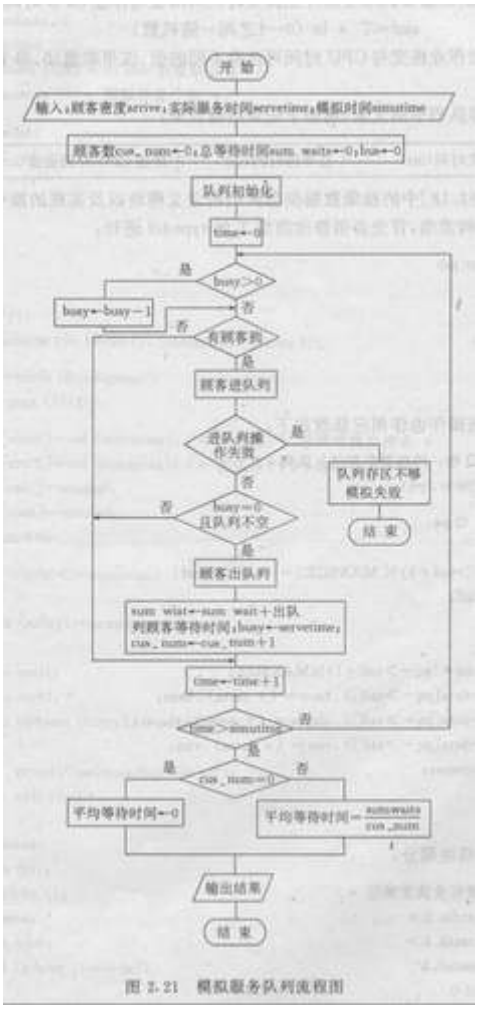


图 2.21 模拟服务队列流程图

计算机操作系统中的作业调度也通过队列来实现，下面是一个简单的模拟。

[例 2。20]操作系统作业调度模拟。

[解]程序采用"事件驱动方式"工作，共有三类事件：

• 作业提交:作业进等待队列,如果 CPU(假定只有一个)不忙,从队列中取出一作业执行.本例中假设有两台不同的终端可提交作业.

• 作业完成:进行规定的统计工作,从队列中取出一新作业执行.

• 时间片用完:为防止一个作业占用 CPU 太久,规定每个作业一交最长占用 CPU 时间,如未完成则重进等待队列(注意有关统计要保证正确完整).

用一个长度为 4 的数组 eTable 存放四个事件(两个终端提交 smt1、smt2,作业完成 cplt,时间片用完 tout)中的每一个到下一次产生间隔的时间单位数，其值要随时间推移而修改，当值为 0 时相应事件发生。与上个例子不一样。不必按单位均匀推进时钟，而是先选出最近一个将发生的事件（在 eTable 诸项中值最小）然后将时钟一下推进到发生时间上，所有与时间有关的信息必须同时修改。

用一个专门的经验公式来生成作业提交时间间隔值与作业 CPU 时间需求值：
val=T * ln (0-1 之间一随机数)

T 为比例因子，对作业提交与 CPU 时间可选取不同的值，这里取值 15。每个作业时间片长取值为 20。

每个作业（即队列中的元素）用如下记录结构表示：

job

提交时间 (int)	已等待时间(int)	所需 CPU 时间余值 (int)

程序引用[例 2。18]中的抽象数据类型队列的定义模块以及实现的操作模块。但由于队列类型已改为结构类型，首先必须修改类型 T 的 typedef 语句：

```
typedef struct job
{
int base;
int elapse;
int run;
}T;
```

另处，进队列操作也作相应修改如下：

```
/*函数 enQ 将一结构型数据送入队列*/
status enQ (term,pq)
T * term;Q pq;
{
if(((pq->tail+1)%MASIZE)= =pq->head)
return fail;
else
{
pq->tail=( +1)%MASIZE;
(pq->data[pq->tail]).base=(*term).base
(pq->data[pq->tail]).elapse=(*term)elapse;
(pq->data[pq->tail]).run=(*term).run;
}
}
```

以下是应用模块部分：

```
/*操作系统作业调度模拟*/
#include <stdil.h>
#include <math.h>
#include "queuel.h"
```

```
#define smt1 0
#define smt2 1
#define cplt 2
#define tout 3
#define exax 4
#define smtmean 15
#define cpumean 15
#define cpulimit 20
#define unused 1000/*eTable 表项缺省值*/
#define simunum 50/*模拟作业个数*/

Qzone jobqzone;
Int clock =0,eTable[emax],count=0;
Ling tobalqt=0;
T curr;
Q jqueue;
Main()
{
int entry1;
stqtus jobsmt ();jobfin (),jobstart (),jobreq();
jqueue=initQ (&jobqzone);
srand (rand()%20);
eTable[smt1]=val (smtmean);/*生成第 1 个 1 号终端提交作业*/
eTable[smt2]=val (smtmean);/*生成第 1 个 2 号终端提交作业*/
eTable[cplt]=unused;
eTable[tout]=unused;
curr.run=0;
while (count<=simunum)
{
switch (entry1=nextevent())
{
case smt1;
case smt2;
if (jobsmt (entry1)= =fail)
{
printf("\nsimulating fails\n");
exit( );
}
break;
case cplt;
jobfin();
break;
case tout;
if (jobreq()= =fail)
```

```
{
printf("\nsimulating fails \n" );
exit();
}
}
}
printf("%-10s%-10s%s\n","","Total","Average");
printf ("%-10s%-101d%5.2f\n","CPU Q:",totalqt,totalqt/(float)count);
}
/*函数确定下一个该发生的事件*/
int nextevent()
{
int I,event =0;
for (I=0;I<emax;I++)
if (eTable[i]<eTable[event])
event =I;
newtime (eTable[event]) /*按该事件时间修改时钟*/
return event ;
}
/*函数 newitme 修改时钟及 eTable 中各项值*/
return event ;
}
int I;
for (I=0;I<emax; eTable;[I+ +]-=timesection);
return clock +=timesection;
}
/*函数 jobsmt 处理作业提交,产生下一同类事件时间*/
status jobsmt (event)
int event;
{
T temp ;
Status jobstart();
Temp.base=clock;
Temp.elapse=0;
Temp.run=val(cpumean);
If (enQ(&temp,jqueue)= =fail)
Retur fail;
Printf("%5d Submitting job \n ",clock);
ETable[event]=val(smtmean);/*产生下一同类事件时间*/
Return jobstart ();/*试启动一作业*/
}
/*函数 val 利用随机函数产生所需时间值*/
int val (mean)
int mean;
```

```
{
returnj (int)(-mean *log (rand()%10000/10000.0));
}
/*函数 jobstart 处理作业启动,产生 eTable 中时间片一作业完成项*/
status jobstart()
{
T * temp;
If (curr.run!=0)
Return success;/*当前作业未到时间*/
If (emptyQ(jqueue))
Return success;/*无等待作业*/temp =deQ(jqueue);
Curr.run=temp->run;
Curr.elapse =temp->elapse+clock-temp->base;
ETable[tout]=cpulimit;
Etable[cplt]=temp->run;
Printf("%5d Starting job \n",clock);
Return success;
}
/*函数 jobfin 处理作业完成统计,启动新作业*/
status jobfin()
{
status jobstar();
eTable[tout ]=uused;
eTable[cplt ]=uused;
count++;
totalqt+=curr.elapse;
curr.run=0;
printf("%5d Finishing job \n",clock);
return jobstart ();
}
/*函数 jobreq 处理时间片完事件,重新入队列*/
status jobreq()
{
eTable [tout] =unused;
eTable [cplt]=unused;
if (enQ(&cur,jqueue)= =fail)
return fail;
curr.run =0;
printf("%5d Requeueing job \n ",clock);
return jobstart ();
}
```

2.4 栈

2.4.1 栈的特得和基本操作

在讨论栈之前,先看一个单间的例题.

[例 2.21] 回文串识别.

输入一字符串,判断它是否为一回文串.的谓回文串是指去掉其中的空格一标点符号等非字母符号后,从前后两个方向读到的串相同,例如:

Ten animals I slam in a net .(我将十个动物装在网里)

[解]对上节讨论的队列结构上的操作稍加改动,便能很方便地解决这一问题.将出队列操作实现为如下函数:

```
/*函数实现双向出队列操作*/
T * deQ (pq,tag)
Qpq;int tag;
{
T * temp;
If (pq->tail!=pq->head)
If (tag= =0)
{
pq->head=(pq->head+1)%MASIZE;
temp =& (pq->data[pq->head]);
return temp;
}
else
{
temp=& (pq ->data[pq->tail]);
if (pq->tail= =0)
pq ->tail =MAXSIZE-1;
else pq ->tail=pq->tail-1;
return temp;
}
}
```

原先的出队列只允许队列首部取出元素,改动后允许从两端取出,用参数 tag 指示取出端.如此改动后,回文判断可实现如下;

```
/*判断回文章*/
#include<stdio.h>
#include "queue.h"
```

```
main ()
{
Qzone qstrzone;
Q qstr;
Char c;
Qstr=initQ(&qstrzone);
Printf("\n a char?");
While((c=getchar())!="\n")
If (((c>64)& &(c<91))‖((c>96)&&(c<123)))
If (enQ(&c,qstr)= =fail)
{
prinf("\nSting too long\n");
exit();
}
while (!(EMPTYq(qstr)))
{
c=* deQ (qstr,0);/*tag 为 0,取队列首元素*/
if (empyQ(qstr))
break;
if (c!= *deQ(qstr,1))/*tag 为 1,取队列未元素*/
{
printf("nIsn,a palindrome.\n");
exit();
}
]
printf("\nIs a palindrome .\n");
}
```

本例中只以队列允许出口作了改动,对算法设计影响非常明显.回文判断实现得极为简单,只不过是一对对地取字符比较而已.实际上这一点改动已在原来的结构中加入了栈的特征.

栈与队列有许多相似之处:都是线性连续的数据结构;都可以随意插入取出元素(理论上大小均可在非负范围内任意变化);进入栈的元素和队列中一样,次序也不能再改变;其访问方式也都有是受限的;其差别在于队列出、入口分别为首、尾两端，而栈出入口是同一端，被称为栈顶，结果使得任何时刻取出的元素总是取近入栈的元素。因此栈称为后进先出（LIFO）结构。
类似于队列，栈有如下基本操作：

- 创建(CreateSTK);
- 初使化(InitSTK)：按约定置栈为状态;
- 数据进栈(push)：在栈顶加入一新数据项

• 数据出栈(pop)：从栈顶取出一数据项；

• 栈空(EmptySTK)；判断是否栈中无数据项

实现中必须定义栈满状态，因此有相应的操作：栈满判断是否已满。

也可以定义只查看但不取出栈顶数据的操作：查看栈顶(ExamSTK)

2.4.2 栈结构的实现

本节中只讨论用静态数组类型实现栈的一种方法。

在对栈操作过程中，由于数据是"后进先出"，栈底位置（即向空栈中插入第一落千丈数据项时所占用的位置）总保持不动，因此不会产生队列那样需移动数据的问题。与队列相比，栈的实现要简单一些。

[例 2. 22]用静态数组实现结构。

[解]假设元素类型为 T，估计栈中元素最大个数为 MAXSIZE，栈可表示为如下结构：



约定 top（尽管称为"栈顶指针"）指向真正栈顶位置下面一个空单元，即新数据将插入的位置。初始状态下 top 指向 STKdata[0]这时栈处于"空状态"。

关于抽象数据类型栈的各种说明放在文件 stack.h 中.

下面模块中的 T 暂取为整型。

```
/*抽象数据类型为整型*/
#define MAXSIZE 100
typedef int T;
typedef struct
{
T stkdata [MAXSIZE];
Int top;
}
STKzone;
Typedef STKzone * STK;
Typedef enum{true=1,false=0}bool;
Typedef enum{success ,fail}status;
```

```
Extern STK initSTK();
Extern status push ();
Extern status pop();
Extern status exam ();
Extern bool full STK ();
Extern bool emptySTK ();
```

与静态实现的队列一样，应用模块必须显式定义 STKzone 类型变量，以建立栈的实际存区。

栈基本操作由下列模块实现：

```
/*栈操作实现模块*/
*/stack.c*/
/*函数 initSTK 执行栈初始化，建栈指针*/
STK initsSTK(stack-zone)
STKzone * stack-zone;
{
STK p;
P=stack-zone;
p->top=0;
}
/*函数 push 将元 tem 送入栈*/
status push (term,pstk)
T * term ;STK pstk;
{
if (pstk->top= =MAXSIZE)
return fail;
pstk ->stkdata[pstk->top]= * term;
(pstk->top)+ +;
return success;
}
/*函数 exam 查看栈顶值，但不取出元素*/
statuss exam (pdata,pstk)
T * pdata ; STK pstk ;
{
bool emptySTK();
if (emptySTK(pstk))
return fail;
pdata =pstk->stkdata[pstk->top-1];
return success;
}
/*emptySTK 判别栈是否空*/
bool emptySTK (pstk)
```

```
STK pstk;
{
returnpstk->top= =0);
}
/*函数 fullSTK 判别栈是否满*/
bool fullSTK (pstk)
STK pstk;
{
return (pstk->top= =MAXSIZE);
}
```

2.4.3 栈的应用

栈结构在计算机语言处理中起着重要的作用，下面两个经简化了的例子可以使读者了解一种表达式处理技术的基本思想。

[例 2. 23]计算机用运符后置法表示的表达式的值。

[解]通常表达式用中置法表示，即二元运算符总是写在两个运算分量当中；这种表示方法本身不能指出求值过程中的运算顺序，必须依赖关于运算符优先级与结合律等背景知识才能正确计算，而且还常常要使用括号。用后置法则可完全避免使用括号，也不再依赖优先级等概念，计算顺序完全由表达式形式所确定。

例如，表达式（a+b*c）/d-e 用后置法表示应为 abc*+d/e-.

对后置法表示的表达式求值按如下如下规则进行：自左向右扫描，每遇到一个 n+1 元组（opd1,opd2…,opdn,opr）(其中诸 opd 为操作数，opr 为元运算符)，就计算一次 opr(opd1, …opdn)值，其结果取代原来表达中 n+1 元组的位置，再从表达式开头式开头重复上述过程，直到表达式中含有运算符为止。

上述过程可以方便地用栈结构实现，为突出与数据结构有关的问题，在本例中只考虑四则算术运算，且假设输入的操作数均为 1 位十进制数（0-9），并且输入的后置形式表达式不含语法错。

设置一个操作数栈，输入字符若为数字则进栈；若为运算符则连续两次取栈顶元素，进行相应计算。C 语言程序如下：

```
/*后置法表达式求值*/
#include<stdio.h>
#include"stack.h"
#define add 43/*四种运算符*/
#define subs 45
#define mult 42
#define div 47
STKzone expSTK zone;
```

```
STK expSTK;
Int opd1,opd2,temp,c1,eval();
Void synerror();"
ExpSTK=initSTK (&texpSTKzone) ;
Printf(`\npost exprssion´);
Whine( (c=getchar())!='\n)
{
if(c= =' ')
continue;
if((c>47)&&((c<58))
{
prinft(''\nexpression too long\n'');
exit();
}
}
else if ((c= =add)||(c= =subs)||(c= =mult)||(c= =div))
{
putchar(c);
if(pop(&opd1,expSTK)= =fail)
synerror();
temp=eval(c,opd2,opd1);
push(&temp.expSTK);
}
else
synerror();
}
if(pop(&opd1,expSTK)= =fail)
synerror();
if(!(emptySTK(expSTK)))
syerror();
printf(''\=%-3d\n'',opd1);
}
/*函数 synerror 处理出错情况*/
void synerror()
{
prinft(''/nexpression ryntax error'');
exit();
}
/*函数 eval 进行算术运算*/
int eval(tag,a1,a2)
char tag;int a1,a2
{
whitch add:
return (a1+a2);
```

```
case subs:(a1-a2);
case mult:
return(a1*a2);
case div:
return(a1/a2);
}
}
```

[例 2．24]将一个运算符中置表达式转换为运算符后置表达式。

[解] 假设输入为不超守 80 个字符，一次从终端读入缓冲区 buffer 内。

比较表达方式的两种方法可知,操作数出现的顺序不变,运算符处理则较为复杂。假定只允许使用下列非数字字符：″+″、″-″、″*″、″、″、（″、″）″。在后置表达式中，先出现的运算符先使用，因此在中置表达中读到运算符并能确定是否该输出，要与后而后运算符作先级比较，若优先级高于栈顶元素则进栈，否则输出栈顶元素，然为″（″，那么读到任何运算符以及″（″都应当进栈，换句话说，″（″在输入流中的优先级与它在栈内的优级不相同，函数 stack prec 与 in prec 分别返回各非数值符号在栈中与输入流保的优先数：

c	+	—	*	/	(	$
stack_prec(c)	1	1	2	2	0	—1
in prec(c)	1	1	2	2	3	

$是开始时放入栈底的特殊符号，这样可以避免单独处理″栈空″状态，它的优先级最纸。注意，″）″设有优先数，一旦读到″）″，栈一直退到第一个遇到的″）″出栈
为止（如没有″（″则表达式语法有错）。

图 2。22-图 2。24 所示为主要算法流程。

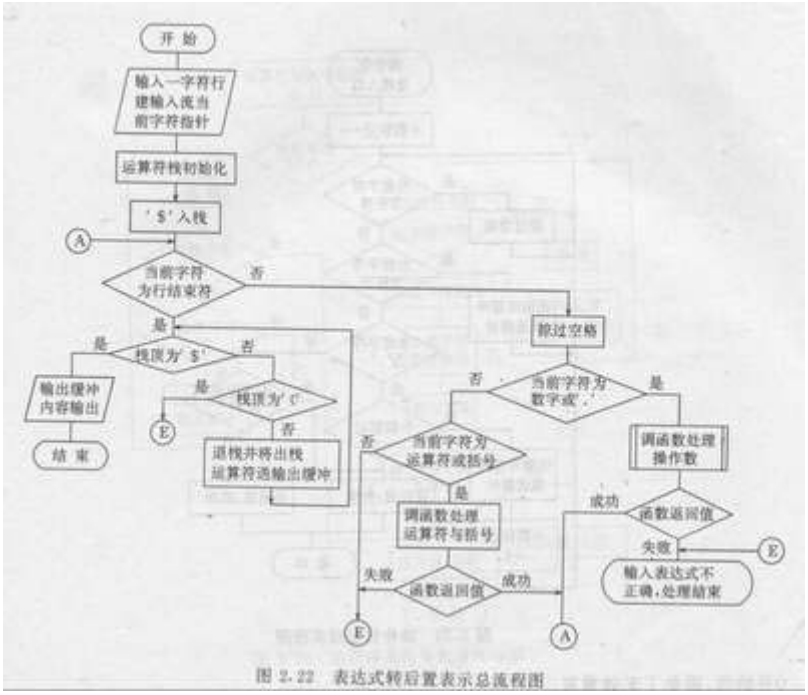


图 2.22 表达式转后置表示总流程图

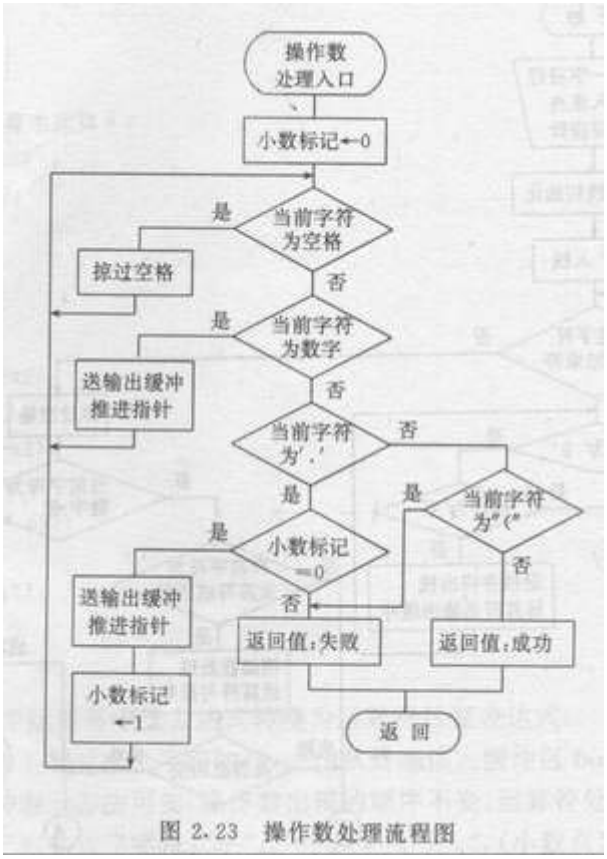


图 2.23 操作数处理流程图

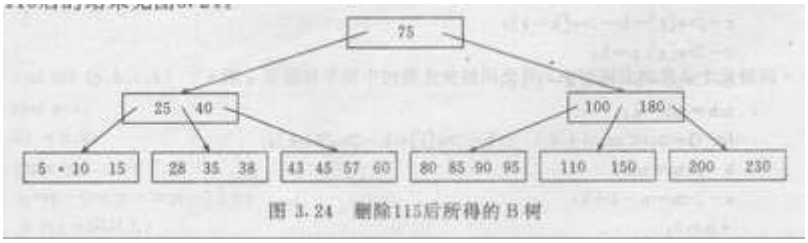


图 3.24 删除115后所得的B树

回溯是一种探性求解技术，在程序设计中经常使用，下面是一个用栈结构实现的例子。

[例 2．25]迷宫问题。

走迷宫是一种广为人知的古老游戏，让老鼠走迷宫也是一个经典心理学实验。在计算机中用一个 M×N 数组表示迷宫，0 表示可通过，1 表示不可通过。

[解] 每个位置上，可选择如下 8 个方向之一：每个方向给一个编号。如图中所示。设当前位置为（I，j），则下一个允许位置应该为在迷宫范围以内，位置为：（I+k，j+l）（k，l=-1，0，+1）的 0 值位置。为省去判别是否越出边界的负担，在迷宫外围加一圈元素，值均为 1，因此对于 M×N 的迷宫。使用的数组有（M+2）行、（N+2 列）

设定入口为左上角元素，出口为右下角元素。走迷宫采用回溯技术，在每一个有多出路的位置上，采用约定的策略选择一个出路。如果尚未到出口，遇到无进路的位置，则应返回最近的一个可用的分支点，重新选择出路。

设置一个栈保存已经过多的分支位置信息，注意，只有一条路可以经过的位置不必记录。在那里不存在重选的可能性，这样可以加快回溯的速度。

在每一个位置上，处理方法如下：若当前位置已经是出口则成功。否则计算当前位置出口数，已经走过的位置（如来路）应排除，将经过的位置值改为非 0。（假如程序结束前想恢复迷宫原貌，应置于 1 的值）。若出口数为 0 需回溯，即取出栈顶元素作为当前位置信息，栈空表明此迷宫无解；若无口数为 1 无可选择，直接推进；若出口数大于 1。则将当前位置信息送入栈中。

这里称的"当前位置信息"除其行、列号外还包括一个方向值，约定找出路时总从 0 方向开始顺时钟进行，在栈内保存了每个分支点上已经选过的方向最大值，这样回溯时不必再从 0 开始选，避免了无效重复。

这样表示迷宫中的一个位置，也就是栈中一个元素的数据应有如下结构，它用来取代上一节中栈定义模块中类型 T 定义语句：

```
typedef struct
{
int I,j,dir;
}T;
```

移动位置采用[例 2.15]中走马的方法,用两个一维数组 move、movej 保存 0-7 方向上相应的行、列号增量。当前位置为（cur i ， cur j）。

由于栈元素改为结构类型，栈操作实现模块吸几个函数也有相应改动，包括与栈有数据交换的 push、poh 与 exam。

```
/*函数 push 将一结构型数据送入栈中*/
staus push(term,pstk)
T*term;STKpstk;
{
if (pstk-＞top= =MAXSIZE)
return fail;
(pstk-＞stkdata[pstk-＞top]).i=(*term).i;
(pstk-＞stkdata[pstk-＞top]).i=(*term).j;
(pstk-＞stkdata[pstk-＞top]).i=(*term).dir;
(pstk-＞top)++;
return success;
}
/*函数 pop 从栈中取出一结构型数据*/
status pop (pdata ,pstk)
T*pdata;STKpstk;
{
bool emptySTK'pstk();
if (emptySTK(pstk))
return fail;
(pstk-＞top)--;
(*pdata).i=(pstk-＞stkdata[pstk-＞top]).i=(*term).i;
(*pdata) .j=(pstk-＞stkdata[pstk-＞top]).i=(*term).j;
(*pdata) .dir=(pstk-＞stkdata[pstk-＞top]).i=(*term).dir;
return success;
}
/*函数 exam 将栈数据送指定位置，但不退栈*/
staus exam(pdada,pstk)
T*pdata;STKpstk;
{
bool emptySTK();
if (emptySTK(pstk))
return fail;
(*pdata).i=(pstk-＞stkdata[pstk-＞top]).i=(*term).i;
(*pdata) .j=(pstk-＞stkdata[pstk-＞top]).i=(*term).j;
(*pdata) .dir=(pstk-＞stkdata[pstk-＞top]).i=(*term).dir;
return success;
}
```

用 C 语言编写的迷宫程序如下：

```
/* 走迷宫 */
#unckude＜stdio.h＞
#unckude''stackl.h''
#define rows 13
#define columns17
staric int maze[rows][columns]=/*迷宫*/
{{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, , },
{1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1},
{1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1},
{1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1},
{1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1},
{1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1},
{1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1},
{1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1},
{1,1,1,0,0,0,1,1,0,1,1,0,0,0,0,0,1,}
{1,0,0,1,1,1,1,0,0,0,1,1,1,1,0,1,}
{1,0,1,0,0,1,1,1,1,1,0,1,1,1,1,0,1}
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
static int movei[8]={-1,-1,0,1,1,1,0,-1};
static int movej[8]={0,1,1,1,0,-1,-1,-1};
STKzone backt zone;
STK backtr ;
Int cur I=1,cur j=1,d=2 nexti,nextj,nextd[2];
Main()
{
bool stopmaze=false;
status back(),fork();
void printout();
backtr=initSTK(&backtr zone);
while(!stopmaze)
{
switch(wayout(cur I,cur j,d))
{
case -1:
stopmaze=true;
breakl
case 0:
}
if (fork(cur j,nextd[1]= =fail)
printf(''\nno solution\n'');
exit();
}
break;
```

```
case 1:
cur I=nexti;
cur j=nextj;
d=0;
maze[cur i][cur j]= ''*'';
}
}
printout();
}
/*函数 wayout 返回出口个数*/
int wayout (I,j.dd)
int I,j,did;
{
int count,k,t1,t2;
for(k=dd,count=0;(k＜=7&&count＜＝1);k+ +)
{
if((I= =d(rows-2))&&(j= =(columns-2)))
return -1;
if (maze[I+movei[k]][j+movej[k]]= =0)
{
nexd[count+ +]=k;
if (count= =1)
{
nexti=I+movei[k];
nextj=j+movej[k];
}
}
}
return count;
}
/*函数 back 执行回溯操作*/
status back (ip.jp,dp)
int*ip,*jp*dp;
{
T temp;
If(emptySTK(backtr))
Return fail;
Pop(&temp,backtr);
*ip=temp.i;
*jp=temp.j;
*dp=temp.dir;
return ruccess;
}
/*函数 fork 将分支点信息送入栈内*/
```

```
status fork (I,j,d)
int I,j,d;
{
Ttemp;
Temp.i=I
Temp.j=j;
Temp.dir=d;
Return(push(&temp.backtr));
}
/*函数 printout 打印结果*/
void printout()
{
int I,j;
printf(''\n'') ;
for (I=0;i＜rows;I+ +)
{
for(j=0;j＜conumns;j+ +)
if(maze[i][j]= =' * ')
printf(''%c'', ' * ');
else
printf('' %d'',maze[i][j]);
printf('' \n'');
}
}
```

2.5 集合

2．5．1 集合的特征

集合是数字中最基本的概念，也是程序设备语言中常用概念。例如，将数据类型看作量组值的集合和在上可施予的操作的集合（见 2。1。1 节）。

在程序设计语言中，为了描述一些离散事件的状态可用集合。例如，在电梯中有 8 个人，各人要停的档层是随机的，如有人到第 6 层、有 3 人到第 18 层、有 2 人到第 37 层、有 4 人到第 60 层，电梯应分别在第 6、48、37、60 层停，描述电梯在哪些楼层停的变量可看成是集合类型，记这个变量为 st 则这个变量 st 取的值是{6，18，37，60}。而{6，18，37，60}是 1-60 这 60 个整数集合的一个子集。变量 st 能取这个整数集合的任一子集，因此 st 的类型--集合类型的值集是这个整数集合的幂集（即子集集合。例如{1，2，3}的幂集为{{}，{1}，{2}，{3}，{1，2}，{1，3}，{2，3}，{1，2，3}}）。在这个例子中，将各人要停的楼层作为一个离散事件。"离散事件"涉及很广泛，例如，一部机器盍里程碑部件的状态都可作离散事件，各部件的故障是随机的，检测机器是否能继续运转也可用集合来描述，若对每个部件编号，在 3、5 号部件有故障时，可用{3、5}表示。达到

一定数量故障或一些关键件的故障便应停机。有了"机器能否运转"的集合类型变量，由计算机判定是否应停机也就十分方便了。

2．5．2 集合结构的实现

C 语言中没有直接提供集合类型，但可用字位串来实现集合类型，即将字位串的每一位对应一个离散事件。例如，前述电梯"在哪层楼停"变量可看成是一个 60 位的字符串，在第 6、18、37、60 层停则是字位串上第 6、18、37、60 为 1，其余位数为 0。将字位串中每一位对应集合的一个元素，无意之中集合元素就有次序了，这是程序设计中引入集合与数学中不一致之处；另一处不一致之处是数学集合的元素只谈相异而无其它约束，而程序设计中集合元素的类型要求是相容的。

字位串的实现可以用字符数组或用整数数组。

关于集合的运算如数学中一样，有集合的并、差、交、相等、不等、包含、蕴含和属于等运算，而它们的实现也是用字位逻辑运算符来完成的。例如，可以定义如下集合基本操作：

·建一个集合，并置其空集（initS）

·插入一个元素(insertS)

·判断元素是否在集合中(memberS)

·判断集合是否为空(emptyS)

·在集合的操作中还可以包括数学上常用的集合运算：

·求两个集合的并集（unionS）

·求两个集合的交集(interS)

·求两个集合的差集(diffS)

·判断子集关系（subS）

判断集合相等（equalS）

[例 2．26]用字位串实现抽象数据类型集合。

[解]假定集合内元素类型为 T，T 的值集大小为 maxsize，而且存在一个从 T 到自然数集的子集{1，2…，maxsize}上的一一对应的映谢，则可以用一个 maxsize 位的字位串表示集合，该字位串第 I 相应的 T 类型元素在集合中。

把任意类型 T 映射到｛1，2…，maxsize｝上要由应用模块完成。本节讨论中假定集合可能的元素即为｛1，2…，maxsize｝的元素。

大多数语言中不提供字位类型。例如在 C 语言中，字位数量少的类型是字符型（8 位）。这里就用字符类型来处理字位串，实际上是用字符型数组实现集合结构。

当 maxsize 给定后，所需字符数组的大小应为[（maxsize+7）/8]。

采用静态方式实现集合，应当在应用模块中定义集合变量的存区（字符型数组）。使用的集合型变量实际上是指向存区的指针。

因为字符已经是最小可寻址单位，对字位的操作通过预先设置的 8 个常数｛128，64，32，16，8，4，2，1｝来进行，注意这 8 个数二进这恰好都只有 1 位是 1。

集合结构说明模块如下：

```
/*抽象数据类型集合定义模块 set.h */
#define maxsize 64
#define len(maxsize+7)/8
typedef char setzone [len];
typedef enum {success,fail} status;
typedef enum  {true=1,false=0} bool;
static char bit[8]= {128,64,32,16,8,4,2,1} ;
extern set inits();
extern status insertS();
extern status0 deleteS();
extern bool memberS
exten boolemptyS();
extern void unionS
extern void interS();
extern void diffS();
extern bool subS();
extern bool equalS();
```

集合操作实现模块如下：

```
/*抽象数据类型集合操作定义模块*/
/*set.c*/
#include "set.h"
/*bbz 涵 initS 进行集合初始化*/set inits (a)
setzone * a;
 {
int I;
char * p
p=a;
```

```
for(I=0;i＜len;I+ +)
p[i]=′ \0′ ;
return a;
}
/*函数 inserts 将元素 e 加入集合中*/
status inserts(e,s)
int e;set s;
{
char*p;
p=s;
if(e＜0｜｜e＞＝maxsize)
return fail;
p[e/8]=bit[e%8];
return success;
}
/函数 deletes 从集合中删去元素 e*/
status deletes (e, s)
int e; set s;
{
char*p;
p=s;
if(e＜0｜｜e＞＝maxsize)
return fail;
p[e/8]&=bit[e%8];
return success;
}

/*函数 members 判断元素是否在集合中。*/
bool members (e,s)
int e; set s;
{
char * p;
p=s;
if (e<0‖e>=maxsize)
return false;
return(p[e/8]&(bit[e%8]));
}
/*函数 emptyS 判断 s 是否为空集*/
bool emptyS(s)
set s;
{
int I; char * p;
p=s;
for (I=0; I<len; I++)
```

```
if (p[i]!=0)
return false;
return ture;
}
```

其中操作用类似方法实现也很简单，这里不再详细列出。

用字位串表示集合，操作实现方便，效率也很高，但必须先完成从元素类型值集到{1，2，3，
… N}上的映射。

下面简单介绍一下实现集合类型的其它方法。

从集合通常的书面表示{X1，X2，…，Xn}看，用数组表示集合是很自然的，这种方法对任何元素类型均适用，尤其是元素类型值集很大，但集合中通常包含元素不多的问题可采用数组实现方法。用数组实现集合操作，开销远大于用字位串。

另外一种实现方法是 Hash 函数法，它适用于元素类型值集很大，但一般使用的集合量其元素个数不超过某个不太大的定值 m 的情况。主要思想是根据问题背景找一个适当的多对一映射期循集元素映射到集合（1，2，…,m）上，建一个数组表示这个集合，这个问题的详细讨论已超出本章的范围。

2．5．3 集合结构的应用

集合结构在计算机算法设计中应用很多，且与排序、搜索等问题密切相关，这些问题本书后面有专门章节讨论。本节举两个直接涉及集合概念的例子。

【例 2．27】 用筛法求素数。

〔解〕筛法源于古希腊，所谓"筛"就是集合。假定要计算 1000 以内的素数（应当适当修改上节中集合定义模块中的 maxsiz，），开始时 1000 以内全部自然数均在筛中，删除 1 后，从 2 开始，由小到大，每次"筛出"留在筛中的最小自然数，同时"筛去"它的所有倍数；则每次筛出 的那个最小自然数均是素数。

此算法速度过慢，并不真正用于数论程长中，不过它是了解集合类型用法的好例于。以下为 C 语言程序：

```
/*筛法求素数*/
#include <stdio.h>
#include "set.h"
setzone sievezone;
set sieve;
main()
{
int I,j,next;
```

```
set init();
status insertS(),deletes();
bool emptyS(), members();
sieve=initS(siebezone);
for (I=1;I<=maxsize;I++) /*集合置初值*/
inserts(I,sieve);
deletes((next=1),sieve);
while(!emptyS(sieve))
{while(!memberS (next,sieve))
next++;
printf("%d\n",next);
j=next;
while(j<=maxsize) /*删除 next 的倍数*/
{
deletes(j,sieve);
j+=next;
}
}
}
```

集合结构常用于文字处理。下面一个例子选自 1987 年程序员级水平考试中一道 Pascal 语言试题。

[例 2·28] 正文统计。逐个字符读一段正文，当读到全部 26 个大写字母或正文结束则处理结束，要求输出读过的总字符数，各大写字母首次出现在字符流中位置序号，最先出现的 5 个大写字母各自出现频度。

[解]假定正文已放在文件 test－text 中，用集合 cap 作为大写字母控制集合，其初值
含全部 26 个大写字母，读到的字母均从 caP 中删除。集合 first5 记录前 5 个出现的大写字母，
其初值为空。两个一维数组 order 与 occur 分别记录各大写字母首次出现的序号与出现频
度。

各大写字母到集合门{1,2，…，26} 的映射由函数 maP 实现。

以下为 C 语言程序：

```
/*利用集合结构处理大写字母*/
# include＜stdio. h＞
#include"set. h"
setzone capzone, flrstzone;
set cap, flrst5
FILE * text;
```

```
ststlC int order [27] ={0};
StstlC iit occur [27] ={0};

main ()
{
char c;
int count=0, rcount=0, k, I;
cap =inits (capzone) ;
rirst5=inits (rirstzone) ;
if ( (text=fopen ( " test sext 。txt, " " r " ) ) = =NULL)
{
printf ( " \nant't open file TEXT\n " ) ;
exit();
}
c= 'A ';
for (k=1;k<=26;k+ +/*集合 cap 置初始：含全部字母*/
inserts (map (c+ +) , cap) ;
while ( ( (c=getc (text) ) ! =EOF) && (! emptys (cap) ) )
{
count+ +;
if (members ( (k=map (c) ) , cap) ) /*第一次读到一字*/
{
dcletes (k, cap) ;
order[k]=count;
if (fcount (5) /*属最早读到的 5 个大写字母*/
{
inserts (k, first5) ;
occur[k]+=1p;
fcount+ +;
}
}
else if (members (k, first5) /*修改频道*/
occur[k]+=1;
}
printf ( " \nTotal of characters: " %d\n " ,count) ;
printf( " \nLetter First occurrence Occur number\n " );
for(c=' A' :c<=' z' :c+ +)
{
printf( " \n%-13c " ,c);
if((I=order[(k=map(c))])= =0)
continue;
printf( " %-16d " ,i);
if((I=occur[k])= =0rde[k]= =0)
continue;
```

```
printf(" %-4d ",I;)
/*函数 map 将大写字母映射到{1，2…，26}上*/
int map（c）
char e;
{
return（（（c＞64&&（c＜91））？（c-64）：（-1））；
}
```

2.6 串

2．6．1 串的特征与基本操作·

串是数组结构的一种特例，其元素均属于一个指定的集合。通常说的串指计算机可表示字符构成的字符串。

与一般数组变化，串作为一个整体的值更为重要，称为串内容。串长长度由内容决定，随着串值的变化而变化。

正由于串的元素类型的特殊性以及从应用角度关心其内容甚于关心单个元素值，使串上的操作与一般数组结构上的操作大不相同。串上的基本操作包括：

·创建一个串（createStr）：新创建的串不含字字符，长度为 0；

·确定串长（strlen）确定当前串上字符数；

·并置（Constr）：将两个串并于一个；

·插入（insstr）：将一个串插入另一个串中的指定位置；

·删除（delestr）：从串中指定位置起删去指定长度的字符段；

·子串（substr）：从串中指定位置开始复制出指定长度的串；

·搜索（earchstr）：从指定位置开始，在串中找出与一指定串内容相同的子串；

·比较（compstr）：比较两个串的大小，通常按字典顺序。

2．6．2 串的实现

字符串是计算机中使用最广泛的一种结构，每种语言都采用各自的方式对字符串结构提供实现支持。许多语言中提供一组串处理运行子程序。

本节中假定除输入输出外没有其它直接支持手段可用。

实现串的存储结构有两种选择：静态与动态。本节中只讨论用静态数组实现串。

[例 2. 29]用静态数组实现抽象数据类型串。

[解] 假定串最大允许长度为 80，为了能确定实际长度，每个串值以字符 nul（ASCII 码 0）结束，串不会出现 nul。

为使实现串操作的子程序说明与操作对象长度无关,用指向串首地址的字符型指针作为操作对象,这样这些实现函数若改用动态结构将不影响使用这些函数的应用模块。

为简单计，不考虑操作数为串常用（如"abcd"等）的情况。以下为串结构实现模块：

```
/*串结构定义模块*/
/*str type.h*/
#define maxsize 80
#define str end' \0'
typedef char strzone[maxsyze];
typedef char*sir
typedef enum {fail.success}status;
typedef enum{true=1,false=0}bool;
str initstr();
int strlen();
status insstr();
status delestr();
status substr();
int searchstr();
int compstr();
/*串操作实现模块*/
/*str type.c*/
#include " str type.h "
/*函数 initstr 进行串初始化*/
str initstr(st)
strzone*st;
{
char*temp;
temp=st;
*temp=str end;
return temp;
}
/*函数 Strlen 给出串长度*/
int strlen (s)
strs;
{
int I;
for (I=0; s[i]! =str end; I+ +) ;
```

```
return I;
}
/*函数 constr 将 S1，S2 并为一个串*/
status constr（s1，s2）
str s1，s2;
{
int I，j，k;
if（（I=strlen（s1））+（j=strlen（s2））＞=maxsize）
return fail;
for（k=0；k＜＝j；k+ +）
s1[I+k]=s2[k];
return success;
}
/*函数 insstr 将 S2 插入 S1 中指定位置*/
status insstr（s1，n，s2）
str s1，s2；int n;
{
int I，j，k;
if（（n＜0 | |（n＞（I=strlen（s1））））| |
（I+（j=strlen（s2））＞=maxsize））
return fail;
for（k=n；k＜j；k+ +）
s1[k+j]=s1[k];
for（k=0；k＜1；k+ +）
s1[n+k]=s2[k];
return success;
}
/*函数 delestr 从 s1 中指定位置起删 n2 个字符*/
status delestr（s1，n1，n2）
str s1 ；int n1，n2;
{
int I，k;
if（（n1＜0）| |（n1＞（I=strlen（s1））））
return fail;
for（k=0；k＜n2）&&（s1[n1+k]）！=str end；k+ +
s2[k]=s1[n1+k];
s2[k]=str end；return success;
}
/*函数 searchStr 从 s1 中位置 n 起搜索子串 s2*/
int searchStr(s1,s2,n)
str s1,s2;int n;
{
int 12,I;
strzone empzoe;
```

```
str temp;
str initStr();
status subStr ();
if ((12=Str Len(s2)= =0)
return n;
temp=initStr(&tempzone);
for (I=n;s1[1+12]!=str _end ;I+ +)
{
stuStr(temp,s2)= =0)
return -1;
}
int m,k,11,12;
m=((11=StrLen(s1))<=(12=StrLen(s2)))?11:12;
for (k=0;k<=m;k+ +)
if (s1[k]<s2[k])
return-1;
else if(sl[k]＞s2[k])
return 1;
return 0;
}
```

2.6.3 不回溯的字符串搜索算法

上一节中实现的搜索操作算法非常直观，从 s1 的第一个字符开始，逐个比较，S1 的指针经常需要回溯。例如：在"abababc"中搜索" abc"，当比较到第 3 个字符发现不匹配时， s1 指针退到第 2 个字符再与"abc"中的 a 比较，显然效率低。

[例 2. 30] 实现不回溯的字符串搜索。

[解] 使用 knuth morris pratt 算法。先考察钱下什么情况下需回溯。假定在字符串 s1 中收索字符串 s2：

s1=" abcbabcbd…" s2=" abcb abcd "

▲* * ▲*

扫描到*时发现不匹配，此时回溯到▲以前的任何位置，实际上必然匹配失败，只有回溯到▲才有可能匹配,这是由于 s2 中从▲开始存在一子串与 s2 首部若干字符相同，这样也不必回溯了，原先到*是比较到第 8 个字符，现在扫描位置仍停留在*，只看作是比较到第 4 个字符，即前 3 个字符已匹配，现在用 s1 中的*与 s2 中的*相比。显然只要能记住 s2 中首部模式重复出现，就可以避免回溯，不过当前字符可能需要比较一次以上。注意，通常搜索问题中 s2 比 s1 短得多，因此只需用较小的代价对 s2 进行预处理，可以避免在 s1 中多次回溯，由此可提高效率。

设置一个一维数组 next，其元素个数与 s2 长度与 s2 长度相等，用于记录 s2 是的模式。next[0]恒为-1，在未遇到与 s2[0]相同字符前，相应的 next 诸项均为

0，若 s2[k]=s2[0]。则 next[k]=next[0]=-1。从 s2[k]起，如果有 t 个字符与 s2[0]，s2[1] ，---s2[t-1]依次相等。而第 t+1 个符不相等。则 next[k+t]=t，即在与 s1 匹配时，若有一子串匹配到 s2 中第 k+t 个字符才发现不等，则从 next[k+t]中取出 t 作为已匹配字符个数。

算法 C 语言描述如下：

```
/*Knuth Morris Pratt 字符串搜索算法*/
void preproc(s2,next)/*函数 preproc 建 s2 模式表*/
str s2;
int next[];
{
int I;j
j=next[0]=-1;
do{
if(j= =(-1)||s2[i]= =s2[j])
{
I+ +;j+ +;
Next[i]=(s2[j]= =s2[i])?next[j]:j;
}
else j=next[j];
}
while(s2[i]!=str end);
}
int searchstrl(s1,s2,n)
str s1,s2;int n;
{
int exit{maxsize},j,k;
void prepric();
if(s2[0]= =str end)
rerurn 0;
preproc(s2,next);
k=n;
for(j=0;s1[k]!=str end;)
{
k+ +;j+ +;
if (s2[j]==str end)
return(k-j);
}
return(-1);
}
```

习题

习题 2。1-2。10 中各含一段程序，在每个空格内按习题说明，各填入一个恰当的表达式或语句。

2．1 从矩形窗口看线段。

[说明]给出一个窗口与一条线段，要求判断通过该窗口能否看到线段的全部或部分；若只能见到一部分，要给出能见部分的两个端点。

线段用两个点构成的数组表示。矩形窗口则用三元组分别给出左下角点位置以及高与宽。

判断的依据是线段与矩形四边形交点的个数，以及线段两端点中有多少个矩形窗口内。

假定已定义下列两个函数：

·intersec(wp,lp,a):计算机 lp 所指的线段与 WP 所指窗口四边相交信息。四个值按上、右、下、左顺序放到数组 a 中。对上、下边交战位置表示为它们与窗口左下角横坐标差，对左、右边则是相应的纵坐标差。若一边上无交点，相应位置-1。

·in(wp,pp):判断 所指的点是否在 wp 所指窗口内，返回值为 1/0（是/否）。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * */
/* prob 1 LOOKING THTOUGH A WINDOW */
/* * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
double x,y;
}
point;
typedef struct
{
point base;
double height,width;
}
wdw;
typedef piont lineseg[2];
void intersec();
int in();
int window view(wp,lp result)
wdw *wp;lineseg lp,result;
```

```
{
double a[4];
int I,(A);
intersec(wp,lp,a);
if(in(wp,lp[0])&&in(wp,lp[1]))
(B);
for(I=o;(c);I+ +)
if(in(wp,lp[i]))
{
result[i].x=lp[i].x;
result[i].y=lp[i].y;
tag=1;
}
for(I=0;(D);I+ +)
{
if(a[i]!=(-1))
{
restult[tag].=wp-＞base.x+
(E);
result[tag].y=wp-＞base.y+
tag=1;
}
}
return(G);
```

2.2 生成单偶阶魔方阵。.

[说明] n 是单偶数，是指 n 是 2 的倍数，但不是 4 的倍数，即 n/2 为奇数。

将单偶阶 n×n 魔方阵分为 4 个（n/2）×（n/2）阶小方阵，每个阶数为奇数。小方阵按下列次序编号。

①	④
③	②

利用本章介绍的算法可以用自然数 1-n 构成四个不同内容奇阶魔方阵，在此基础上通过大主阵中的元素换位得所求结果。换位规则如下：（设 k=（n/2-1）/2）

- 块①中前 k 行与块④对应元素互换，但第 1 行中间元素不动。
- 块①与块④的中心元素互换。
- 块③中最下面 n/2-k-2 行元素与块②中相应元素互换。

假设已定义函数 oddmagic(n，a)，它用从 a 开始连续 n2 个自然数生成一个(奇数) n 阶魔方阵；函数 exchange 完成两个元素互换。

a1，a2 是在函数外定义的数组变量，分别 oddmagic 与 evenmagic 的结果。

```
/ * * * * * * * * * * * * * * * * * * * * * */
/ Prob.2 EVEN MAGIC SQUARE * /
/ * * * * * * * * * * * * * * * * * * * * * */
Void oddmagie();
Void exchange();

Void evenmagic(n)
Int n;
{
int I,j,k,m,s,t,temp;
(A);
for(k=0;k＜3;k+ +)
{
oddmagic(m,(B));
if((c))
t=1;
s=((D))
for(I=0;i＜m;i+ +)
for(j=0;j＜m,j+ +)
a2[I+t*m]=a1[i][j];
}
for(I=0;i＜(m/2);I+ +＝
for(j=0;j＜m,j+ +)
exchange(a2[i][j]),a2[i][j+m]
(E);
for(I=0,i＜(n/2-m/2-3;i+ +)
for(I=0;j＜m;j+ +)
exchange(a2[n-I-1][j],a2[n-I-1][j+m]);
}
```

2. 3 生成拉丁方阵。

[说明] n×n 拉丁方阵的每行、每列均为自然数 1，2，…,n 的一个全排列，每行（列）上均无重复数字。

下面的例子说明生成得法

```
1 4 3 2 5
5 3 2 1 4
4 2 1 5 3
```

3 1 5 4 2
2 5 4 3 1

第一行元素用随机产生。从 1 开始依次将 1，2，…,n 填入第一行，填入的列号由随机函数产生。建一集合 s，初值为空，用它来判别列号是否重复。

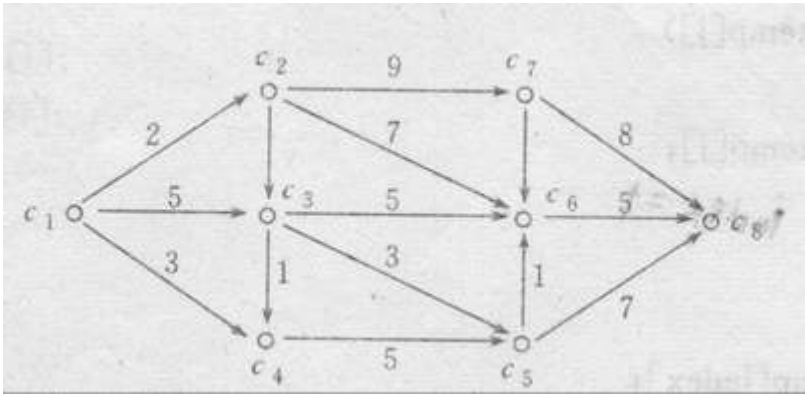
第一行完成后，即以第一行作为全方阵索引，即若第一行中第 I 个元素值为 j，则 j 在各行中列号即为在第一行中从位置 I 开始读（到行末则循环）到 n 个自然数。例如第一行第 4 个元素为 2，则从 4 开始读出"4、3、2、5、1"即元素 2 在各行中列号（注意这是从 1 计数的列号）。

假设有本章介绍集合设施可用。

```
/* * * * * * * * * * * * * * * * */
/*prob 3LATIN SQUARE */
/* * * * * * * * * * * * * * * * */
set s;
viod latin square(n)
int n;
{
int I,j,sub,index;
for(I=1;i＜=n;I+ +)
{
do index=rand()%n;
while(member(index,s);
(A)
a[0][index]=I;
}
for(I=0,i＜n;I+ +)
{
for(j=0;j＜n;j+ +)
if((B))
break;
sub=(C);
for(j=I;j＜n;j+ +)
a[j][(D)]]=a[0][i]
}
}
```

2.4 最短路径问题。

[说明]用有和图表示道路网，有向边上的权表示两点间距离，要对下图示出从 c1 到各点最短路，可用 Dijdstra 算法。



对每个顶点加一标号（数字或无穷大）。标号分临时、永久两种。临时标号表示从 c1 到该点距离的上界，永久标号即最短路径长。

临时标号按下述方法改永久标号：

对每个已有永久标号的点 ci 检查所有从 ci 有一有向边可达而又带临时标号的点 cj 临时标号改为

min{cj 原标号，ci 永久标号+有向边 cicj 的权}然后在上述全部 ci 中选出标号最小的改为永久标号。

开始有时只有 c1 有永久标号，诸（cj（j≠1）标号均取无穷大。

数组 a 为有向图邻接矩阵，TEMP 与 PERM 分别记录临时标号与永久标号，值-1 表示一点无该类标号。为简单计，下列函数 shortest path 不记录最短路径轨迹。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * /
/* Prob 4 THE SHORTEST PATH */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
void shortest path(n,a,perm)
int n,a[][100],perm[];
{
int temp[100],I,j.count,low ,index;
for(I=0;i＜n;I+ +)
{
perm[i]=-1;
temp[i]=65535;
}
perm[0]=0;
for(count=1;count＜n;count+ +)
{
low=(A);
index=-1;
for(I=0;i＜n;I+ +)
```

```
if ((B))
for(j=1;j＜n;j+ +)
{
if((a[i][j]＞)&&((C))
&&(temp[j]＞perm[i]+a[i][j]) ;
temp[j]=perm[i]+a[i][j];
{
low＞temp[j]
{
low=temp[j];

( D )
}
}
perm[index]=temp[index];
((E))
}
```

2.5 改变区域颜色。

[说明] 一个象点方阵构成彩色图形区，任意指定一个起始点坐标（I，j），要求将该点所在的同色四连区域全部改为一种颜色 new。四连区域指该区域中任一点至少在上、下、左、右四方向之一与同区域其它点相邻。仅是对角相邻的不属该区域。

颜色用整形数表示。用一个栈来控制区域的延伸。为处理方便，有效象点方阵外围包一圈值为-1 一元素（所有合法颜色均为正值）。

函数中利用本章介绍的栈设施。

```
/* * * * * * * * * * * * * * * * * * * */
/*prob 5 CHANGE THE COLOR OF A REGION */
/* * * * * * * * * * * * * * * * * * * */
typedef struct
{
int x,y
}pixel;
STK colorSTK;
Void change (a,I,j.new)
Int I,j,new,a[][SIZE];
{
pioel curr;
int(A),(B ),k,old;
curr.x=I;
curr.y=j;
```

```
old=a[i][j];
push(curr,colorSTK);
while((C));
{
pop(&(curr),colorSTK);
((D));
for(k=0;k＜4 ；k+ +)
{
if(a[curr.x+xcor[i]][curr.y+ycor[i]]= =old)
{
curr.x+=xcor[i];
curr.y+=ycor[i];
(E)'
}
}
}
}
```

2.6 回文算术

[说明] 任给一个三位数 abc（x 表示一个十进制数字），算出 a b 与 c b a 之和。若该和数不是回文数（即从左边向右读与从右边向左读是同一个数，如 19391，）再按上述方法求和。以此类推，直到得到回文形式的和数位数已超过 15 位时中止计算。

用字符串表示数。函数中利用本章中的队列与栈设施。函数 output 输出和数。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * */
/*Prob6 ARITHMATIC WITH PALINDROME */
/* * * * * * * * * * * * * * * * * * * * * * * * * /

void output();
int palindrome();
int palin arith();
{
int I,count,a,b,c,sum[20];
STKzone palinSTKzone;STK pxlinSTK;
Qzone palinQzone;Qpalin;
(A);
(B);
for(I=0;i＜3;I+ +)
sum[i]=rand()%10;
sum[3]=-1;
count=1;
while(!(phlindrome(sum)))
```

```
{
output(sum);
for(I=0;sum[i]!=-1,(C);I+ +)
{
push(sum[i],palinSTK);
enQ(sum[i],plinQ);
}
if((D))return 0;
c=0;
I=0;
While((E));
{
POP(&a,palinSTK);
DeQ(&b,palinQ);
(F);
sum[I+ +]=(a+b+c)%10;
}
if(c＞0)
(G)=c
sum[i]=-1;
(H);
}
output(sum);
returncount;

}
```

2．7 地图着色问题。

[说明]任何平面区域图均可用四种颜色着色，使相邻区域颜色互异，这就是著名的四色定理。

要求对给定的区域图，排出全部可能的四着色方案（包含用色少于四种的可能方法）。

区域相邻关系用矩阵表示，每个区域有一个序号（从 0 连续排列）。adj[i][j]=1 表示区域 I 与 j 相邻，数组 color 记录每个区域已着上的颜色（用 1，2，3，4 表示）。

利用本章中的栈结构实现回溯。一种着色方案完成时，采用强制回溯以实现输出全部可能方案的要求。

以下程序段中包含四个函数：

·coloring:对所给邻接矩阵 adj 找出全部着色方案;

·select:对区域 I 在指定的几种颜色范围内选出可用颜色，或报告失败（返回 -1）;

·colorOK:判断对区域 i 指定颜色 c 是否可用;

·back:选色失败时或完成一种着色方案.

另外假定已定义函数 output,每次输出一种着色方案.

```
/* * * * * * * * * * * * * * * * * * * * * * * */
/Prob7 FOUR COLROORING A MAP */
/* * * * * * * * * * * * * * * * * * * * * * * /
void output();
STKcolorSTK;
Int back(ip,color)
Int *ip,color[];
{
int c=4
while(c= =4)
{
if(emptySTK(colorSTK))
return 0;
pop(&c,colorSTK);
(A);
(B);
}
return c;
}
int colorOK(I,c,adj,color)
int I,c,acj[][100],color[];
{
int j;
for(j=0;(C);J+ +)
if(D)
return 0;
return 1;
}
int select(I,c,acj,color)
int I,c,adj[][100],color[];
{
int j,k;
for(k=c;k＜＝4 ; K + + )
if(colorOK(I,k,adj,color))
```

```
return 0;
}
void coloring(n,adj)
int n,adj[[100]];
{
int color[100],I=0,c=0;
while(1)
{
if((c=select(I,c+1))= =0)
{
if((c=back(&I,color))= =0)
(E);
}
else
{
push(c,colorSTK);
color[I+ +]=c;
c=(G);
}
else
c=(H);
}
}
}
}
```

2.8 医务室模拟。

[说明] 假设一个医生，在一个小时内随机地来几位病人，每个病人所需处理时间为 1-9 分钟之间某个随机值。为简单计，假定上述值均已形成，放在数组 a 中 。

每个病人用结构 patient 表示，arrive 为到达医务时间，treat 为处理时间，假定 a 中元素已按到达时间先后排好序，clock 为时钟。

以下函数描述主要模拟部分：

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*Prob8 DOC TOR SERVING SIMULATION */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
int arrive,treat;
}patient;
QpatientQ;
```

```
Void clinic(n,a,average wait,doctor wait)
Partient a[];
Int n,*doctor wait;
Float average wait;
{
patient curr ,*pp;
int wait=0,dwait=0,I=0,k,clock=0;
for(k=0;k＜n;k+ +)
{
if(empty(patientQ);
(C);
(D);
wait+=clock curr,arrive;
while((E))
{
curr.treat-=(a[i],arrive clock);
enQ(a[i],patientQ);
clock=a[i],arrive;
}
(F)
}
*average wait =wait/(float)n;
*doctor wait=dwait;
}
```

2. 9 找欧拉回路。

[说明] 假设输入是正确的欧拉图，用矩阵表示，元素（i，j）值为整型数 k，表示顶点 I 与 j 之间有 k 条无向边（k≥0）。显然正确输入必为对称矩阵。

由图论知识可知，所求的回路必可由若干个无公共边的子回路构成，且全部子回路可以连在一起。

算法大致如下：从顶点 0 出发，逐步选相邻点，每经过一条边，则将它从图中删除，一旦遇到起始点则完成一子回路，记入二维数组 path 的一行中。若图中还有非孤立点，选择一点重复上述过程，直到图中不含边为止。然后全并各子回路，从 path 的第一个元素开始，按行逐个元素送输出一个元素，判断它是否为某个子回路起始点（注意，这里所谓"回路的起点"是针对程序执行过程而言的）。若是则转去先输出新子回路。这实际上是个递归的过程，因此用栈结构实现。

下列程序中含五个函数：

• euler trail：建立子回路表以完成整个欧拉回路；

• next：选择下一个顶点；

• unfin：判断是否还有边未走过；

• merge：合并各子回路；

• noewcircle：转而输出一新子回路。

```
/* * * * * * * * * * * * * * * * * * * * * * * */
/*Prob9EULERIAN TRALL */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
int row,col;
}pos;
int graph[25][25],path[25][25],out[300];
STKsubSTK;
Int next(vertex,n)
Int vertex,n;
{
int k;
for(k=0;k<n;k+ +)
if(graph[vertex][k]!=0)
{
(A);
(B);
return(-1);
}
int unfin(n)
int n;
{
int i,t;
for (i=0;i<n;i+ +)
if(C)
return-1;
}
int newcircle(pi,pj,m)
int pi,pj,m;
{
int I;
pos temp;
for(i=0;i<m,(D);i+ +)
if(E)
{
if(path[pi][pj]!=path[pi][0])
{
```

```
temp.row=pi;
temp.col=pj+1;
push(temp,subSTK);
}
return I;
}
return(-1);
}
void merge(sub num)
int sub unm;
{
int pi=0,pj=0,pk=0,i,j;
pos temp;
while(1)
{
out[pk+ +]=path[pi][pj];
if(F)
{
pi=t;
pj=1;
}
else
{
if(path[pi][pj]= =path[[pi][0])
{
if(G)
return;
pop(&temp,subSTK);
pi=temp.row;
pj=temp.col;
}
else
(H);
}
}
}
void euler trail(n)
int n;
{
int vertex=0,pi=0,pj=0,i;
for(; ;)
{
path[pi][pj]=vertex;
if(I)
```

```
break;
pi+ +
pj=0;
}
}
}
merge(pi+1);
}
```

2.10 找最大的公共子串。

[说明]输入两个字符串，要求输出最长的公共子串/如果达到最大长度的子串多于 1 个，应全部输出。空串不作为公共子串，函数返回值为最大公共子串个数，其长度送入变量 subl 中。所用串设施如本章所定义。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
*Prob 10LARGEST COMMON SUBSTRING */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
int commstr(str1,str2,subl)
int subl;
str strl,str2;
{
str sh,lo,temp;
int count=0,1,k;
strzone tempzone;
temp=initstr (tempzone);
if(strLen(stfl)＞strLen(str2))
{
sh=str2;
lo=str1;
}
else
{
sh=strl;
lo=str2;
}
for(A);(B);1- -)
{
for(k=0;(c);k+ +)
{
substr(sh,k,1temp);
if(starchstr(to.temp,0)!=(-1))
{
output(temp);
```

```
(D);
}
}
if(count>0)
break;
}
(E);
return count;
}
```

2. 11 本题是对[例 2.9]中图 2.4（a）与（b）按表 2.2 表与 2.3 表算法的两个程序。对应图 2.4（a）的程序（表 2.2 算法）：

```
#include<stdio.h>
#define size 10
int a[size][size];
main()
{int i,j,k,n;
pritntf("\n input n\n");
scanf("%d",&n);
k=1;
for(A)
{if((n+i)%2= =1)
{for(B)
{a[i+j][j]=k;k+ +;
}
}
else
{for(C)
{a[i+j][j]=k;k+ +;
}
}
}
for(D)
{if(((n+i)%2= =1)
{for(E)
{a[j][j-i]=k;k++;
}
}
else
{for (f)
{a[j][j-i]=k;k++;
}
}
}
```

```
printf("\n");
for(I=0;I<=n;I++)
{ for (j=0; j<=n; j++ );
printf("%3d",a[i][j]);
printf("\n");
}
}
```

对应图 2.4（b）的程序（表 2.3 算法）：

```
#include<stdio.h>
#define size 10
int a[size] [size];
main()
{int i,j,k,n,up;
printf("\n input n\n");
scanf("%d",&n);
I=n;j=0;a[i][j]=1;up=0;
For(k=2;k<=((n+1)*(n+1);k++)
{if(up= =-1)
{I=I-1;j=j-1;
}
else
{I=I+1;j=j+1;
}
if(I<0)
{(G);(H);up=～up;
}
if (j>n)
{(I);(J); up=～up;
}
if(I>n)
{(K); up=～up;
}
if(j<0)
{(L); up=～up;
}
a[i][j]=k;
}
printf("\n")
for(I=0;I<=n;I++)
{for(j=0;j<=n;j++)
printf("%3d",a[i][j]);
printf("\n");
```

```
}
}
```

第 3 章 动态数据结构及操作

在第 2 章中讨论了数组，记录等基本数据结构，这类结构的主要特点是一旦建立便不再改变，它们是静态的数据结构。在许多应用中，还会遇到更加复杂的结构，这些结构在处理品过程中可以动态地重新组织，因此这些结构称为动态数据结构，本章介绍链表、树和图三种动态结构。

3.1 链表

一个键表由若干个结点组成，这些结点的存储位置可以是连续的，也可以是任意的，根据一个结点中所含指针的个数，链表可分为单链表、双向链表和多重链表等。

3.1.1 单链表

最简单的链表为单链表，单链表中每个结点由两个字段组成，数据字段和指针字段，数据字段表示元素本身的信息，指针字段指明下个元素的位置。单链表常用来表示线性结构，例如：线性表（a1,a2,…,an-1，an）可以用图 3.1 的单链表表示，图中 head 为头指针，它擀出第一个结点的位置，该变量称为指针。

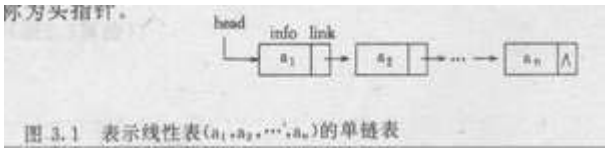


图 3.1 表示线性表($a_1,a_2,\cdots,a_n$)的单链表

大多数高级语言都提供指针类型用来定义链表结构，我们用 C 语言定义链表中的结点结构如下：

```
typedef struct node{
into info;
struct node*link;
}NODE;
```

[例 3.1]依次打印链表中每个结点的 info 字段的值。

[解] 设 head 为链表的头指针，首先从 head 所指的结点开始打印，沿链接的方向向右遍历，直至到达最后一个结点，下面是一个打印链表的递归函数。

```
void lprint ( pointer head )/*-递归打印其 info 内容----*/
```

```
NODE*head
{
if (head!=NULL)
{
printf ("%d \ t,head->info);
laprint(head->link);
}
}
```

在许多问题中，有时为了操作方便，需要在链表的第一个结上噗之前增加一个特殊的结点，称为表头结点，它的 info 字段不含信息或根据问题的需要存放特殊的信息，后面我们会看到有关的例子。图 3.2 是一个带表头结点的单链表

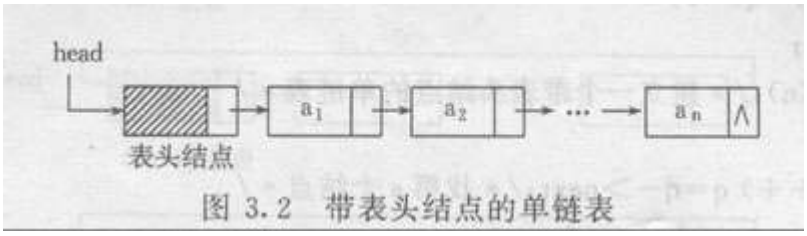


图 3.2 带表头结点的单链表

[例 3.2]建立一个带表头结点的单链表。

[解]设有 n 个元素的一个线性表（a1,a2,…，an），我们先建立一个表头结点，然后再依次建立与 n 个元素对应的 n 个结点，算法描述如下：

```
NODE*creatlinklist (n)
int n;
{int i;
NODE*head,*p,*q;
If (n==0)return(NULL);
head=(NODE* )malloc(sizeof * head);
q=head;
for (i=1;<=n;i++ )
{p=(NODE * ) malloc (sizeof * head);
printf("Enter a element \ n");
p->info=getche ( );
printf ("n") ;
q->mext=p;
q=p;
}
p->next=NULL;
return(head);
}
```

[例 3.3]约瑟夫问题。设有 n 个人坐在圆桌周围，从第 S 个人开始报数，报到 m 的人出列，然后再从焉个人开始报数，数到 m 的人又出列，……，如此重复，直到所有的人都出列为止。要求按出列的先后输出每一个人的信息。

[解] 设圆桌周围的 n 个人的信息构成一个带表头结点的单链表。先找到第 s 个人对应的结点，由此结点开始，顺序打描 m 个结点，将扫描到的第 m 个结点删除，然后复复上述动作，直至输出 n 个结点。下面是实现约琴夫问题的 C 语言程序；

```
typedef char datatype;
typedef struct node {datatype info;
struct node *next;
}NODE;
void joseph (n,s,m)
int n,s ,m;
{int i,j;
NODE *creatlinklist(lint);
NODE*h,*p,*q,*r;
if(n<s)return;
h=creatlinklist(n);/*建立一个带表头结点的单链表*/
q=h;
for(i=1;i<s;i++)q=q->next;/*找第 s 个结点*/
p=q->next;
for(i=1;i<n;i++)
{for(i=1;j<m;j++) /* 报数*/
if(p->next!=NULL)&&(q->next!=NULL)
{q=q->next;
p=h->next;}
elseif(p->next==NULL)
{q=q->next;
p=h->next;}
else{q=h->next;
p=p->next;}
printf("%c \ n",p->info); /*一个元素出列*/
r=p;
if(p->next==NULL)
{p=h->next;
q->next-NULL;}
else{p=p->next;
if ( q->next!=NULL)q->next=p;
else h->next=p;
}
free(r );
}
```

```
printf("%c \n",(h->next)->info);
}
```

3.1.2 循环链表

在单链中，最后一个结点的 link 字段为空（NULL），如果让它的 link 字段指向第一个结点，则构成循环链表，图 3.3 是一个循环链表的示例。

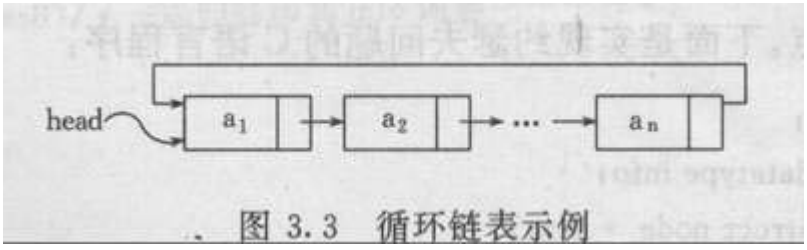


图 3.3 循环链表示例

在图 3.3 中，可以在结点 a1 之前增加一个表头结点，头指针 head 指向表头结点，构成带头头结点的循环链表。图 3.4（a）是一个空表，它仅有一个表头结点，（b）是一个非空表。

在实际应用中，常采用图 3.4(c)循环链表，图中 rear 指向最后一个结点，由 rear 可方便地存取最后一个结点和第一个结点。与单链表相比，循环链表的优点在于从链表中任一结点出发可以访问每一个结点。

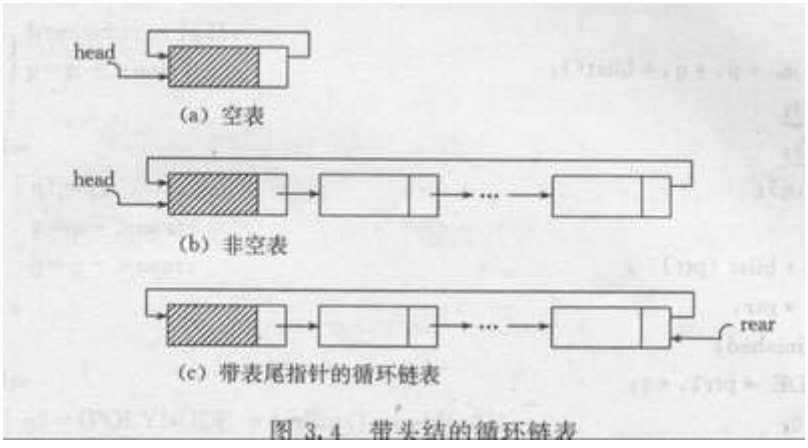


图 3.4 带头结的循环链表

[例 3.4]多项式加法（1990 年程序员试题）。

[解]设有两个变元为 x、y 和 z 的整系数多项式 p 和 q,用循环链表示多项式，链表中每个结点表示多项式中的一项,格式如图 3.5 所示。它表示项 coef·xiyjzk，其中 coef 为项的系数；项的指数 index=I*100+10+k(i,j,k 皆为整数，且 0≤i,k≤9);next 为指向下一个结点的指针。链表中设置一个头结点，头结点的 coef 字段为 0，index 字段为-1。链表中其余结点按 index 值降序排列。例如，多项式 3x6-5x5y2+6y6z 的表示如图 3.5（c）。

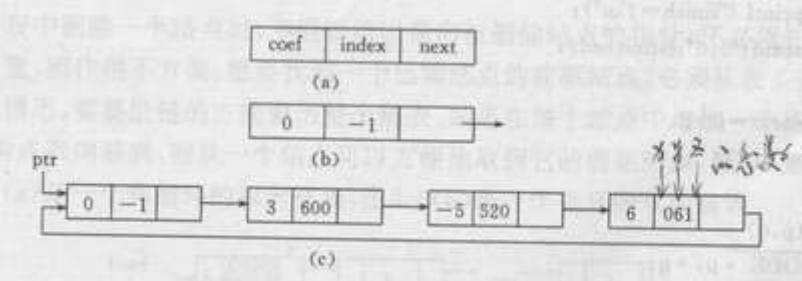


图 3.5 表示多项式的循环链表

下列程序定义两个多项式 p 和 q,通过调用 blist 构成两个循环链表 p*和 q*。将多项式 p 加到 q,相加后，链表 p*不变，q*为两个多项式之和。两个多项式之和。两多项式相加由 polyadd 完成。算法是：比较 p 和 q 的指数，如相等，则系数累加到 q 的对应项系数；如果 p 的项指数小于 q 的项指数，则比较 q 的下一项；如果 p 的项指数大于 q 的项指数，则将 p 的该项插入 q 中。如果系数相加后的值为 0,则从 q 中删除该项。

```
#include"stdio.h"
#include"malloc.h"
typedef struct poly {
int coef;
int index;
struct poly*next;
}POL YNODE;
main( )
{POLYNODE*p, *q, *blist( );
p=bist(p);
q=blist(q);
}
POLYNODE*blist(ptr)
POLYNODE*ptr;
{int i,j,k,finished;
POLYNODE*ptrl,*r;
Finished=0;
Ptr=(POLYNODE*)calloc(1,sizeof(ptr));
Ptr->coef=o
Ptr->index=-1;
Ptr->next=ptr;/*建立头结点*/
r=ptr;
while(finished==0)/*建立多项式各项组成的链表*/
{ptrl=(POLYNODE*)calloc(1,sizeof(ptrl));
scanf("%d%d%d%d", &ptrl->coef,&i,&j,&k);
ptrl->index=i*100+j*10+k;
r->next=ptrl;
```

```
r=ptrl;
printf("finish=?\n");
scanf("%d",&finished);
}
r->next=ptr;
return (ptr);
}
polyadd (p,q)
POLYNODE*p,q;
{POLYNODE*q1,*q2;
p=p->next;
q1=q;
q=q->next;
while (p->index>=0)
{while (p->index<q-index)/*指数相等*/
{q->coef+=q->coef;/*系数相加*/
if(q->coef==0)
{q2=q;
q1->next=q->next;
q=q->next;
free(char * )q2;
p=p->next;
}
else
{q2=(POLYNODE*)calloc(1,sizeof(q2).);
q2->coef=p->coef;
q2->index=p->inedx
q2->next=q;
q1->next=q2
q1=q2
}
}
}
```

3.1.3 双向链表

在单链表中册除一个结点时，不但要给出指向被册除结点的指针，还必须知道该结点前驱结点的位置，操作很不方便。想要找到一个已知结点的前驱结点，必须从表头开始查找。在循环的情形，需要沿链的方向盘遍历整个链表。如果在每个结点中增加工厂一个指向前一个结点的指针，构成双向链表，则人一个结点可以方便地取到它的前驱结点。图 3.6 是双向链表的示例，图 3.6（a）是一个非循环的双向链表，图 3.6（b）是一个循环的双向链表。

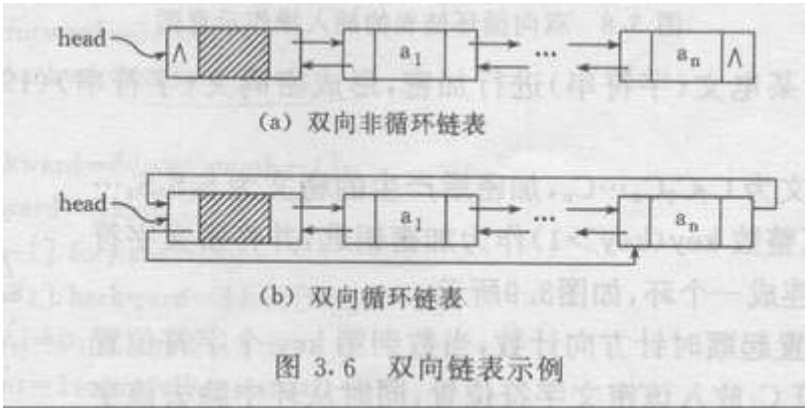


图 3.6 双向链表示例

双向链表结构定义如下：

```
typedef struct node * pointer;
struct node{
datatype info;/数据字段*/
pointer llink,rlink;/*左、右链接*/
}
```

其中,info 为结点本身的信息,llink 和 rlink 分别为指向其前驱和后继的指针。

在双向循环链表中，如果 p 指向链表中某个结点，则有：
p=(p→rlink) →llink=(p→llink) →rlink

这一等式反映了双向链表结构的特点。

在双向链表中，删除一个结点的操作如图 3.7 所示，只需改变该结点的 rlink 和后断结点的 llink，即：
（p→llink）→rlink=p→rlink;
(p→rlink) →llink=p→llink

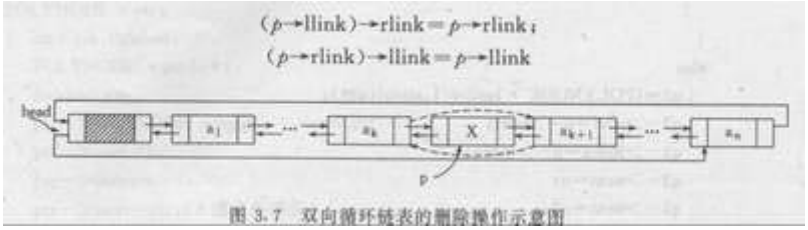
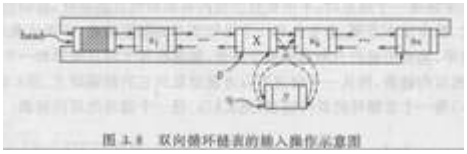


图 3.7 双向循环链表的删除操作示意图

在双向链表中，将一个新结点插在 p 所指的结点之后的操作如图 3.8 所示。设 q 指向半要插入的结点，插入操作的指令如下：
q→rlink=p→rlink
p→rlink=q;
q→llink=p;
(q→rlink) →llink=q



[例 3.5] 对某电文（字符串）进行加密，形成密码文（字符串）（1991 年高级程序员试题）。

[解]假定原文为 C1C2C3…Cn,加密后产生的密文为 S1S2S3…Sn,首先读入一个正整数 key(key>1)作为加密钥匙，并将密文字符位置按顺百针方向连成一个环，如图 3.9 所示。

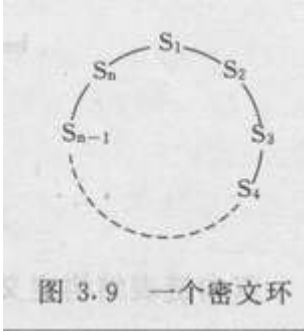


图 3.9 一个密文环

加密时从 S1 位置起顺时针方向计数，当数到第 key 个字符位置时，将原文中的字符 C1 放入该密文字符位置，同时从环中除去该字符位置。接着，从环中下一个字符位置起继续计数，当再次数到第 k 个字符位置时，将原文中 C2 放入其中并从环中除去该字符位置；依此类推，直至 n 个原文字符全部放入密文环中。由此产生的 S1S2…Sn 即为原文的密文。

下列程序中，字符数组 old 存放原文字符，函数 decode 用于将原文 old 加密返回密文字符数组的头指针，密文环用双向循环链表表示。函数 strlen 用于计算一个字符串中的字符个数。

```
#include<stdio.h>
#include<stdio.h>
#defineCR13
typedef struct node
{char ch ;
struct node *forward;
struct node * backward;
}CODE;
main()
{char * decode(),old[256];
int strlen(),key,num=0;
printf("\nPlease input the thlegraph:\n");
old[(num==255)?num:num-1]='\0';
do{printf(\nPlease input key =?(key>1):");
```

```
}
char*decode (old,key)
char*old
int key
{char*new;
int length ,count ,i;
CODE*loop,*p;
Length=strlen (old);
Loop=(CODE*)malloc(length*sizeof(CODE));
For(i=1;i<length-1;i++)
{loop[i].forward=&loop[i+1];
loop[i].backward=&loop[i-1];
}
loop[o].backward=&loop[length-1];
loop[0].forward=&loop[1];
loop[length-1].forward=loop;
loop[length-1].backward=&loop[length-2];
for(p=loop,i=0;i<length;i+ +)
p=p->froward;
p->ch=*old++
p->backward->forward= p->forward;
p->forward->backward= p->backward;
p= p->forward;
}
new=(char * )malloc ((length+1)*sizeof(char));
for(i=0; i<length; i++) new[i]=loop[i].ch;
new[length]=`\0`;
ruturn (new);
}
int strlen(s)
char*s;
{int len =0
while(*s++!`\0`)len++;
reurn(len);
}
```

3.1.4 十字链表和稀疏矩阵

在科学应用中经常需要计算一些很大的矩阵，这些医阵的一个重要特征是大多数远元素都是 0，这类矩阵称为稀疏矩阵。设想一个 1000×1000 的矩阵，它只有 250 个非零元素，如果给该矩阵分配一百万个存储单元，显然是极其浪费的。

一般我们只存储矩阵中的非零元素，稀疏矩阵每一得的非零元素组成一个带表头结点的循环链表，第一列的非零元素也组成一个带表头结点的循环链表。链表中每个结点有五个字段，结点结构如图 3.10 所示。

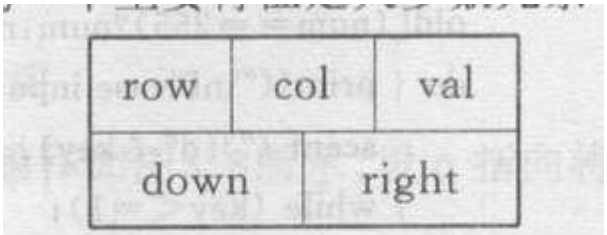


图 3.10 十字链表的结点结构

其中：

row:：指明非零元素所在的行号；

col：指明非零元素所在的列号；

val：非零元素的值；

right：指向同一行中下一个非零元素；

down：指向同一列中下一个非零元素；

由于行链表和列链表成十字交叉，故称十字链表。矩阵 A 的十字链表见图 3.11。

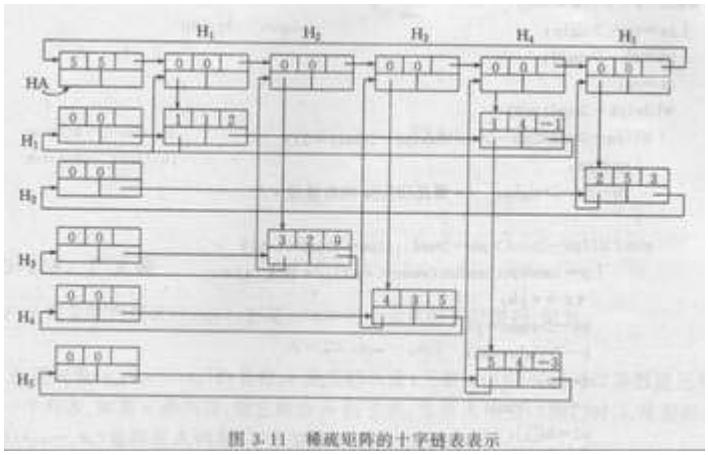


图 3.11 稀疏矩阵的十字链表表示

A=

2 0 0 -1 0
0 0 0 0 3

```
0 9 0  0  0
0 0 5  0  0
0 0 0 -3  0
```

由于每个行链表的表头结点只使用 right 字段，而每个列链表的表头结点只使用 down 字段，所以这两组表头结点可以合用，即对一个 n * n 的矩阵，十字链表中只需要 n 个表头结点。这些表头结束点能过 val 字段链接起来组成一个循环链表，结点 HA 作为整个十字链表的表头结点，它的 row 字段和 col 字段分别表示稀疏矩阵的行数和列数，val 字段指向结点 H1。

[例 3.6] 稀疏矩阵的加法。令 C=A+B，A 和 B 是用十字链表表示的稀疏矩阵，C 为结果矩阵。为简单起见，这里只讨论 A=A+B 的运算。

[解]稀疏矩阵的结点结构定义如下：

```
typedef struct node * nodeptr;
union data{int a;
nodeptr p; };
struct node {int row,col; /*行,列*/
union data val;
nodeptr down,right; };
```

设指针 pa 和 qb 分别指向矩阵 A 和 B 第 i 行上的结点，矩阵运算分下列 四种情况：

（1）若 pa→col==pb→col 且 pa→val+pb→val≠0

（2）若 pa→col==pb→col 且 pa→val+pb→val==0

（3）若 pa→col<pb→col 且 pa→col≠0,pa 向右移动，重新比较。

（4）若 pa→col>pb→col 或 qa→col==0,在 A 的链表插入 bij 的结点。

稀疏矩阵加法程序如下：

```
void sparse-matrix-add(ha,hb)
nodeptr ha,hb;
{nodeptr ca, pa ,p ,pl ,p2,qa,hl[100];
int i=1,j; /*ha 和 hb 的行列是否相等*/
if(ha->row!=hb->row)||(ha->col!=hb->col)return;
ca=(ha->val).p;
while(ca->row= =0)
{hl[i++]=ca;
ca=(ca->val).p; /*数组 hl 赋初值*/
ca=(ha->val).p;
```

```
while(ca->row)==0
{pa=ca->right;
pb=cb->right;
qa=ca;
while(pb->col!=0)
{if(pa->col<pb->col)&&(pa->col!=0)
{qa=pa;
pa=pa->right; /* 情况（3），pa 向右推进*/
}
else{if((pa->col>pb->col)||(pa->col==o))
{p=(nodeptr) malloc (sizeof)(* p);/*情况（4）*/
*p=*pb;
qa->right=p;
p->right=pa;
qa=p;
j=pb->col;
p1=h1[j];
p2=p1
while((p1-<row!=0)&&(p1->row<pa->row))
{p2=p1
q1=q1->down;
}
h1[ j ]=p2;
p->down=q2->down;
p2->down=p;
pb=pb->right;
}
elseif(pa->col==pb->col)
{pa->val.a+=pb->val.a;
if(pa->val.a==0)
{qa->right=pa->right;
j=pa->col; /*情况（2）*/
p1=h1[j];
p2=q1;
while((p1->row!=0)&&(p1->row<qa->row));
{p2=p1;
p1=p1->down;
}
h1[j]=p2;
{p2=p1;
p1=p1->down;
}
h1[j]=p2;
p2->down=pa->down;
```

```
free(pa);
pa=qa->right;
}
else{qa=pa;
pa=pa->right; /*情况（1）*/
}
pb=pb->right;
}
}
}
ca=(ca->val).p;
cb=(cb->val).p;
```

3.1.5 广义表

广义表又称为列表（Lists），它是 n (n≥0)个元素的有限序列，记为

A=（a1,a2,…an）

其中 A 是列表（a1,a2,…，an）的名称，n 是它的长度，元素 a1（1≤i≤n）可以是数据元素，也可以是一个列表。如果 a1 是列表，则它称为 A 的子表。当表 A 不空（n≥1）时，a1 称为列表 A 的头，而（a2,…，an）是列表 A 的尾。下面举几个列表的例子。

（1） D=（ ），D 是空表，它的长度为 0。

（2） A=（a，（b，c）），A 的长度为 2，第一项是数据元素 a,第二项是子表（b,c）。

（3）B=(A,A,( )),B 的长度为 3，前两项是列表 A,第三项是个空表。

（4）C=(a,C),C 是一个递归表，它的长度为 2。C 相当于一个无限的表 C=(a,(a,a…))。

注意(( ))和( )是为同的表,(( ))是长度为 1 的表,它有一个元素( )。而（）是一个空表，它的长度为 0。

由上面的例子可知：一个列表可以作为另外一个列表的子表，它也可以是一个递归的表。列表的结构非常灵活，在 Prolog 和 Lisp 等语言中，列表是语言提供的最重要的和最灵活的数据结构。

列表通常采用链式结构表示，这里我们介绍一和链表结构。在列表中，每个结点由三个域组成:tag,data 和 next，其中 tag 为标志域，用来区别表中的结点是数据元素还是子表。Tag 为零时，表示该结点是数据元素，data 域存放数据元素的

值；否则，data 域存放指向该子表的指针。 next 域存放指向表中下一个元素的指针值。上面所举的四个列表的存储结构如图 3.12 所示。

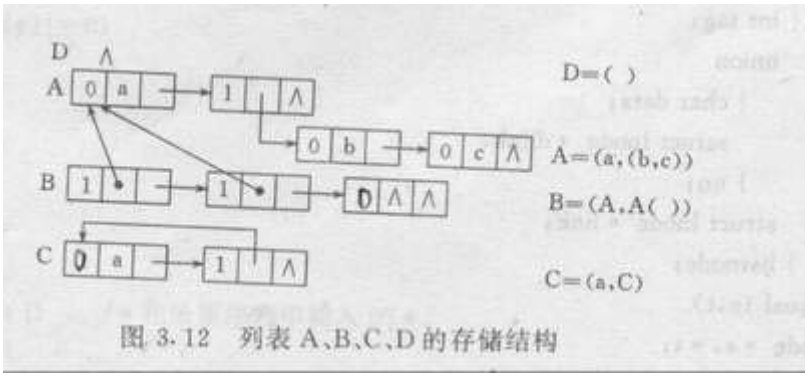


图 3.12 列表 A、B、C、D 的存储结构

列表的结点类型说明如下：

```
typedef struct node*list; /*list 为指向列表结点的指针* /
struct node {int tag; /*标志域*/
union{int data; /*数据域，当 tag=0 时存放数据* /
list sublist;}x; /*当 tag≠0 时存入指向子表的指针* /
list next;}; /*表中下一个元素的指针* /
```

[例 3.7]求列表中数据元素的个数。

[解]列表有以下三种情况：

（1）空表：表中数据元素的个数为 0。

（2）非空表且表头为数据元素：表中数据元素的个数为表尾中数据元素的个数加 1。

（3）非空表且表头为子表：表中数据元素的个数为表头的子表中所含数据的个数加上表尾中数据元素的个数。求列表中的数据中个数的程序如下：

```
elem-num(p) /*计算广义表 P 中数据元素的个数* /
list p;
{if(p= =NULL)return(0); /*空表返回零*/
elseif(p->tag= =0)return(1+elem_num(p->next)); /*表头为数据元素；返回表尾中的个数加 1* /
}
```

[例 3.8]判断两个非递归的列表是否相等（1992 年程序员级试题）。

[解] 设两个列表的首指针分别是 s 和 t，判别方法是：

（1）若 s=t，则表示它们指向同一个列表或两者皆为空表，比时两者相等。

（2）若两者皆非空，比较它们的表头（即表的第一个元素）是否相等以及表尾是否相等，分两种情况：

① 若表头的标志不同。则两列表不等。

② 若表头的标志相同。则比较表头（表头可能也是一个列表，）若表头相等，再比较尾。

比较两个列表的程序如下：

```
#include<alloc.h>
typedef struch lnode
{int tag;
union
{char data;
struct lnode *dlink;
}un;
struct lnode *dlink;
}listnode;
int equal(s,t)
listnode*s,*t;
{int x,res;
if(s= =t) return(1);
else if)(s!=NULL&&t!=NULL)
if(s->tag= = tg->tag)
{if(!s->tag)x=(s->un.data= = t->un.data);
elsex=equal(s->link,t->link));
}
return(0);
```

3.1.6 链表算法例析

[例 3.9] 法雷（Forder） 序列。n 级法雷序列 F。是将分母小于等于 n 的不可约真分数按递增次序排列，并且在第一个分数前加上数 0/1，在最后一个分数后加上数 1/1 所得的序列。例如 F8 为：

0/1，1/8，1/7，1/6，1/5，1/4，2/7，1/3，3/8，2/5，3/7，1/2，4/7，3/5，5/8，2/3，5/7，3/4，4/5，5/6，6/7，7/8，1/1。

它们在数轴上的分布是不规则的。

[解]由于预先无法知道序列元素的个数，所以用链表存放法雷序列。对任一给定的 n，生成一个分母小于等于 n 的不可约真分数，然后把该分数插入序列中，如此重复直到生成有序的法雷序列。程序如下：

```
/*------法雷序列------*/
# include<sttdio.h>
# include<alloc.h>
typedef struct node *link;
struct node{
int numerator. / *分子*/
denorminator; /*分母*/
link next;
};
link fn;
int gcd(int x,int y) / * 求 x 和 y 的最大公约数 * /
{
int rem;
while ( (rem=x%y)!=0)
{
x=y
y=rem;
}
ruturn(y);
}
return(y);
}
void insert(int I ,int j ) /*在法雷序列中插入 j/i*/
{
link p1,p2,p3;
p2=fn;
while (p2->denominator*j>p2->numerator*i)
{
p1=p2
p2=p2->next;
}
p3=(link)malloc(siizeof(*p3));
p3->numerator=j;
p3->denominator=i;
p3->next=p2;
p3->next=p3;
}
/*--------*/
main( )
{
int i,j,n;
link p1,p2;
scanf ("%d,&n");
fn=(link)malloc(sizeof(struct node));
```

```
fn->numerator=0;
fn->denominator=1; /*在序列插入 0/1* /
p1=(link)malloc(sizeof(struct node));
p1->numerator=1;
p1->denominator=1;
p1->denominator=1;
fn->next=p1; /*在序列中插入 1/1*/
for(i=n;i>1;i--)
for(j=1;j<i, j++)
if(gcd(i,j++)printf("--");
do{
p2=p1;
i=0; /*打印序列*/
while((p2!=NULL)&&(i++<10)
{
printf("%d4d",p2->numerator);
p2=p2->next;
}
printf("\n");
for(j=0;j<i; j++)printf("--");
printf("\n")
p2=p1;
i=0
while((p2!=NULL)&&(i++<10))
{
printf("%4d",p2->denominator);
p2=p2->next;
}
printf("\n\n");
p1=p2
}
while(p2!=NULL);
}
```

[例 3.10] 统计职工完成数量并输出名次。（1993 年程序员级试题）

[解]从指定的数据文件中读入职工的工号和他完成产品数量的信息，累计同一职工多次完成的产品个数，最后按以下格式输出；职工完成产品数量的名次，该名次的每位职工完成的产品数量，同一名次的职工人数和他们的职工号（同一名次的职工号按从小到大顺序输出）。

程序采用链表结构存储信息，链表中每个结点对应一位职工。在输入数据的同时形成一个有序链表（按完成的产品数量及工号排序）。在累计某一职工完成的数量时会改变原来链表中结点的顺序。程序如下：

```
# include<stdio.h>
typedef struct ele{
int no,q;
struct ele *next;
}elem;
main ( int argc ,char * *argv )
{FILE*fp;
elem*h*u,*proc( );
if (argc= = 2&&(fp=fopen(argv[1],"r"))!=NULL)
{h=proc (fp);
fclose (fp);
output (h);
while(h!=NULL){u=h->next;
free(h)
h=u }
}
}
elem*proc(FILE*fp)
{int n ,m;
elem*u,*v,*p,*base;
base=NULL;
facanf(fp,"%d",&n);
while (!feof(fp))
{fscanf(fp,"%d",&m);
for(v=base;v!=NULL&&v->no!=n;u=v,v=v->next);
if(v!=NULL)
{if(v= =base)base=v->next;
else u->next=v->next;
v->q+=m;
}
else{v=(elem*)malloc(sizeof(elem));
v->no=n;
v->q=m
}
p=base;
hwile (p!NULL)
if(p->q<v->q||p->q= =v->q&&p->no>n)break;
else{u=p
p=p->next;}
if(p= =base)base=v;
v->next=pl
facanf(fp,"%d",&n);
}
ruturn base;
```

```
}
output (elem*head)
{int count .order;
elem*u,*v;
printf("ORDER QUANTITY COUNT NUMBER\n");
u=head;
order=1;
while(u!NULL)
{for(count=1,v=u->next;v!=NULL&&v->q= =u->q;count++,v=v->next);
printyf("%4d%9d%6d",order,u->q,count)
order+=count;
for(;count- -!=0;printf("%4d",u->no),u=u->next);
printf("\n");
}
}
```

3.2 树

树是一种相当重要的非线性结构，它为计算机应用中出现的具有层次关系和分支关系的数据提供了一种自然的表示，在解决各种算法问题中具有广泛的应用。

3.21 树和二叉的概念

树是一个有限的非空的结点集合 T。其中有一个特定的称为根的结点 r,其余结点分成 m(≥0)个互不相交的集合 T1,T2…Tm,每个集合 T1,又是一棵树,T1,T2…Tm,称为根 r 的子树。

图 3.14 是两棵树的例子。（a）是只有一个根结点的树，（b）是一棵有 13 个结点的树，其中 A 是根，其余结点分成三个互不相交的集合：T1={B，E，K，L}，T2={C，G}，T3={D，H，I，J，M}，它们都是根 A 的子树。同样，我们可以写出 T1，T2 和 T3 的根以及它们的子树。

如果对树中每个结点的子树（如果有的话）从左到右地加以编号，对结点 x,Txi 和 Txj 是 x 的两棵子树,若前者的编号 xi 小于后者的编号 xj,则 Txi 一定在 Txi 一定在 Txj 的左边，该树称为有序树。即诊断一个结点的子树是有次序的，反之，如果不考虑各结点子树的次序，则为无序树。一般都讨论有序树。

一个结点的子树的数目称为这个结点的度。度为 0 的结点称为叶子结点或终端结点。一棵树的度等于树中各结点的度的最大值。树是一种层次结构，我们规定根结点的层次为 1，其它结点的层次等于其父母结点的层次加 1。一棵树中结点的大层次定义为这棵树的高度。图 3.13 中的树的高度的 4。

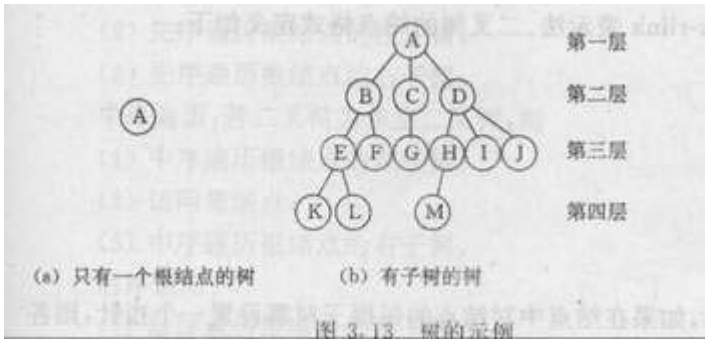


图 3.13 树的示例

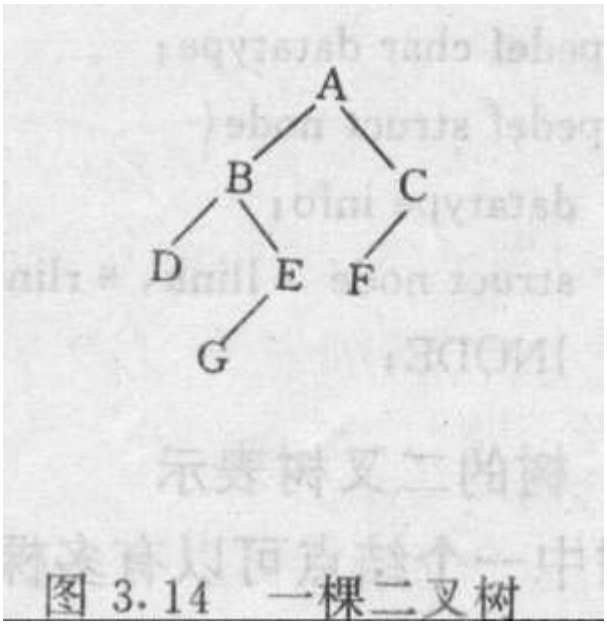


图 3.14 一棵二叉树

森林是 n(n) ≥0)棵树的有序集合，森林与树的概念很相近，对图 3.13（b）中的树，去掉根结点 A 便得到由它的三棵子树组成的森林。

2、二叉树

二叉树是一种特殊的树形结构，二叉树中每个结点的度都不大于 2，二叉树可递归地如下：

二叉树是 n(n)≥0)个结点的有限集合，它或者为空集，或者是由一个根结点加上两棵分别称为左子树或右子树的互不相交的二叉树组成。

图 3.14 是一棵含有 7 个结点的二叉树，二叉树的根为 A，A 的左子树是以结点 B 为根的二叉树；右子树是以 C 为根的二叉树。结点 D、G、F 的左、右子树皆为空二叉树，结点 E 和 C 的右子树为空二叉树。

二叉树第 i 层上最多只能有 2i 个结点，如果一棵二叉树每一层上的结点个数都达到最大值，则该二叉称为满二叉树。一棵高度为 K 的满二叉有 2k+1-1 个结点，图 3.15（a）是满二叉树。

若一棵二叉树至多只有最下面两层上的结点的度可以小于 2，并且最下面一层上的结点都集中在最左边的若干位置上，则此二叉树称为完全二叉树。图 3.15 的（a）和(b)是完全二叉树，而（c）则不是完全二叉树。由图 3.15 的（a）和(b)可以看出，满二叉树一定是完全二叉树。

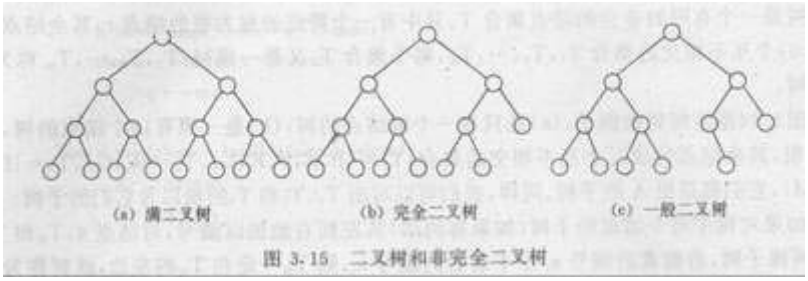


图 3.15 二叉树和非完全二叉树

3.2.2 树和二叉树的存储表示

1、二叉树的存储表示

表示二叉树常采用链接表示，每个结点的格式为

llink	infor	link

其中，info 为结点本身的信息，llink 和 rlink 分别为左子树和右子树根结点的指针。二叉树的这种看示法又称为二叉树的 llint-rlink 表示法。二叉树的结点格式定义如下：

```
typedef char datatype;
typedef struct node{
datatype info;
struct node*llink ,*rlinkl;
}NODE;
```

2、树的二叉树表示

树中一个结点可以有多棵子树，如果在结点中对结点的每棵子树都设置一个指针，则各结点中所含的指针数目不一定相同。树的一种标准存储形式是：若一棵树的度为 d,则每个结点设置 d 个指针。显然，这种表示法将会浪费大量的存储空间。由于树或森林与二叉树之间有一种自然的、一一对应的关系，任何一棵树或一个森林都唯一地对应一棵树或一个森林。我们可以用下列方法将森林转换成对应的二叉树；

（1）将所有兄弟结点连接起来；

（2）对每个有子女的结点，保留与第一个子女的链接，去掉与其它子女的链接；

（3）以根结点为轴心，将树按顺时针方向旋转 45° 。

对图 3.16（a）的树，用上述方法进行转换，行到图 3.16（b）二叉树。

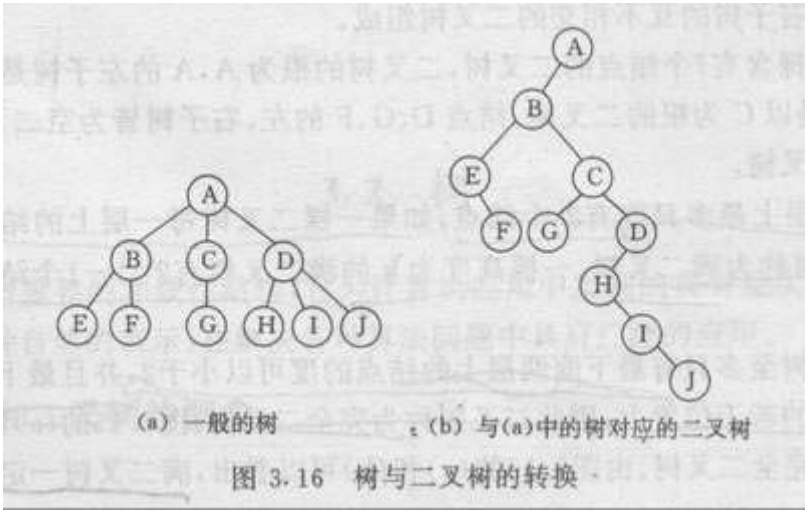


图 3.16 树与二叉树的转换

将森林转换成二叉树的方法是：先将森林中每一棵树转换成二叉树，然后将各二叉树的根结点连结起来。图 3.17（a）是一个森林，（b）是与之对应的二叉树。

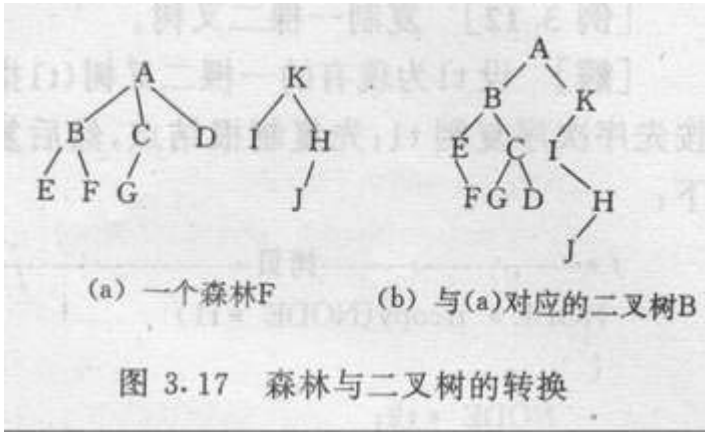


图 3.17 森林与二叉树的转换

3.2.3 二叉树的遍历

二叉树的遍历是指按照一种规定的顺序访问二叉树的每个结点，并且每个结点只被访问一次。二叉树遍历有下列三种方法：

先序遍历：若二叉树为非空二叉树，则

（1）访问根结点；

（2）先序遍历根结点的左子树；

（3）先序遍历结点的右子树。

中序遍历：若二叉树为非空二叉树，则

（1）中序遍历根结点的左子树；

（2）访问根结点；

（3）后序遍历根结点的右子树；

后序遍历：

（1）后序遍历根结点的左子树；

（2）后序遍历根结点的左子树；

（3）访问根结点。

[例 3.11]中序遍历二叉树的算法。

[解]按中序遍历的定义，中序遍历二叉树的递归算法描述如下，算法中 root 指向二叉树根。

```
Void inorder(NODE*roct) /*root 指向二叉树的根*/
{
if(root!=NULL)
{
inorder(root->llink);
printf("\t%c",root->info);
inorder(root->rlink);
}
}
```

[例 3.12]复制一棵二叉树。

[解]设 t1 为现有的一棵二叉树（t1 指向二叉树的根），要得到 t1 的一个付本，我们可以按先序复制 t1:先复制根结点，然后复制根的左子树，再复制根的右子树。递归算法如下：

```
/*------拷贝------*/
NODE*trcopy(NODE*t1)
{
NODE*t2;
If(t1= =NULL)return(NULL);
else{
t2=(NODE*)malloc(sizeof(NODE));
t2->info=t1->info; /*复制根*/
t2->llink=trcopy(t1->llink); /*复制左子树*/
t2->rlink=trcopy(t1->rllink) /*复制右子树*/
```

```
}
}
```

下面是一个等价的非递归算法，算法先建立一个头结点（t2 指向该结点），然后用先序次序复制原有的二叉树，得到一个付本并将它作为头结点的右子树。树遍历 t1 的过程中，若一个结点的右子树非空，则将右子树的根结点推入栈中，以便在遍历左子树后能进入右子树。变量 branch 的值为 0 或 1，用来确定一个新复制的结点应作为父母的左子还是右子女。

```
NODE*copy(NODE*t1)
{
NODE*p,*q,*temp,*t2;
Int branch;
Stack s; /*定义一个栈*/
Push(t2,s);
do{
q=top(s)
pop(s); /*取栈顶结点*/
q=top(s);
branch=1;
while(q!=NULL)
{
temp=(NODE*)malloc (sizeof(NODE));
if(branch= =0)p->llink=temp;
else p->rlink=temp;
temp->info=q->info;
in(q->rlink !=NULL)
{
push(temp,s);
}
else temp->rlink=NULL;
p=temp;
branch=0;
q=q->llink;
}
if(branch= = 0)p->llink=NULL;
else p->rlink=NULL;
}while(!empty STK(s));
temp=t2;
t2=t2->rlink; /*删除头结点*/
free(temp)
return(t2);
}
```

[例 3.13]已知一棵二叉结点的先序序列和中序序列，构造对应的二叉树（1994 年度高级程序员下午试题）。

[解]因为由二叉树结点的先序序列能够唯一地确定一棵二叉树，构造二叉树的方法如下：设欲构造的二叉树的先序序列和中序序列分别为

A，B，D，E，C，F，G
和
D，B，E，A，C，G，F

前序序列中第一个结点 A 为二叉树的根，由中序序列可知，A 前面的结点 D，B 和 E 一定是 A 的左子树上的结点；A 后面的结点 C、G、F 是 A 的右子树上的结点。对 A 的左子树和右子树再分别确定其根结点以及左、右子树上的结点。如此递归地处理，最终可以得到需要的二叉树。

```
/*----94 年高程试题 14 页第 7 题----*/
/*----由前序，中序序列确定二叉数----*/
# include<stdio.h>
# include<stdio.h>
# include<stdio.h>
# define MAX100
typedef struct node {
char info;
struct node*llink ,*rlink;
}TNODE;
char pred[MAX],inod[MAX];
TNODE*restore (char*,int);
Void postorder(TNODE*);
Main(int argc,char**argv)
{
TNODE*root;
if(argc<3)exit(0);
strcopy(pred,argv[1]);
strcopy(pred,argv[2]);
root=restore(pred,inod ,strlen (pred));
postorder(root);
}
TNODE*restore(char*ppos,char*iqos ,int n)/*----建立二叉树----*/
{
TNODE*ptr;
Char*pros;
int k;
if(n<=0)return NULL;
ptr=(TNODE*malloc(sizeof(TNODE));
ptr->info=*ppos;
```

```
for(rpos=ipos;rpos<ipos+n;rpos++)
if(*rpos= = * ppos)break;
k=pros-ipos
ptr->llink=restore(ppos+1,ipos,k);
ptr->llink=restore(ppos+1,ipos,k,rpos+1,n-1-k);
return ptr ;
}
void postorder(TNODE*ptr) /*---后序遍历---*/
TNODE*root;
If(ptr= = NULL)return;
postorder(ptr->llink);
postorder(ptr->rlink);
printf("%c",ptr->info);
}
```

[例 3.14]已知二叉树 root，求与它对应的树的高度。

[解]一棵树的高度应等于根结合各棵子树的高度的最大值加 1。在与树对应的二叉树中，一一个结点的第一个子女（即第一棵子树的根）成为左子女，右兄弟成为右子女。设二叉树采用 llink=rlink 表示法，我们可以写出 下列用 C 语言描述的递归算法。

```
/*-----求树的高度-----*/
int high(NODE* root) /*root 指向二叉树的根*、
{
NODE*P;
Int h,max;
If(root= = NULL)return(0); /*root 指向二树的根*/
{
NODE*p;
Int h,max;
If(root= = NULL)return(0); /*空树，高度为 0*/
P=root;
max=0;
while(p!=NULL)
{h=high(p->llink);
if(h>max)max=h;
p=p->rlink;
}
return(+ +max);
}
```

[例 3.15]已知与森林对应的二叉树，按层次顺序遍历森林。

[解]以图 3.17 中的森林与对应的二叉树为例，二叉树的根结点 A 为森林第一棵树的根，它的右子女 K 为森林中第二棵树的根，这是第一层上的结点。A 的左子女 B 为第二层上第一个结点，森林中一个结点的右兄弟在二叉树中变为右子女，据此我们可以找到 B 的所有兄弟结点（它们处于同一层。）同样也可以找到结点 K 的第一个女子结点 I 及 I 的所有兄弟结点。另外，对森林中同一层上的两个结点 x1 和 x2，若 x1 在 x2 先被访问，则 x1 的所有子女结点必定在 x2 的所有子女结点被访问。下面的算法中使用一个队列来控制结点被访问的顺序，其算法为：

（1）将二叉树的根结点插入队列 Q;

（2）若队列 Q 非空，则访问队头元素对应的结点，并删除队头元素。若被访问的结点的左子树非空，则将其左子女插入队列;

（3）重复第（2）步，直至队列为空。

```
/*----------层次遍历以二叉树表示的森林-----------*/
void leveltraverse(NODE*root)
{
NODE*P
QueueQ; /*Q 为队列*/
P=root
CreatQ(Q); /*建立一个空队列*/;
if(p!=NULL)
EnQ(p,Q); /*将根结点插入队列*/
While(!EmptyQ(Q))
{
p=HeadQ(Q)
{
p=HeadQ(Q); /*取队头元素*/
DeQ(Q); /*删队头元素*/
While(p!=NULL)
EnQ(p->llink,Q);
p=p->rlink;
}
}
}
/*----------------------------*/
```

3.2.4 线索树

对二叉树按某种次序遍历得到一个结点序列，在序列中，若终点 x 在结点 y 之前被访问，则称 x 为 y 的前驱，y 为 x 的后继。如果 x 是在 y 紧前面一个被访问的结点，则 x 为 y 的直接前驱，y 是 x 的直接后继。

对 n 个结点的二叉树采用 llink-rink 表示法，n 个结点中有 n+1 个空链接(空指针)A.J.parlis 和 C。thornton 提出了利用这些空链接来存储一个结点的直接前驱和直接后继的地址的方法：若一个结点的 llink 为空，则使 llink 指向它的直接前驱结点，若 rlink 为空，则使 rlink 指向它的直接前驱结点，若 rlink 为空，则使 rlink 指向它的直接后继结点。指向前驱或后继的指称为线索，带有线索的链接表示的二叉树称为线索二叉树。图 3.18 是一棵中序线索二叉树，其中的虚线为线索。

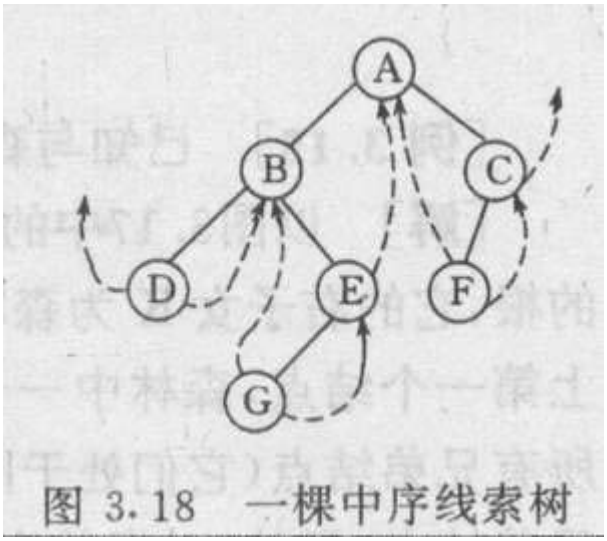


图 3.18 一棵中序线索树

为了区别指向子女的指针和指向前驱或后继的指针，每个结点中增加两个字段 lth 和 rth，线索二叉树中结点格式如下：

lth	llink	info	rlink	rth

其中
lth={0 llink 指向左子女
{1 llink 指向前驱
rth={0 llink 指向右子女
{1 llink 指向后继

用 C 语言定义为：

```
struct node {
char info;
struct node *llink ,rlink;
int lth ,rth;
}
```

利用线索树，找一个结点的前驱点和后继结点十分方便，下面是在一棵中序线索树中找指定结点的前驱和后继的算法。

```
Typedef struct node {
Char info;
Struct node*llink *rlink;
Int 1th rth;
}NODE;
/*--------------指定结点的前驱-------------*/
NODE*inpred(NODE*t)
{
if(t->1th= = 1||t->llink= =NULL)
return(t->llink);
t=t->llink;
while(t->rth= =0)
t=t->rlink;]
return(t );
}
/*------------找指定结点的后继--------------*/
NODE*insucc(to)
NODE*to;
{if(to->rth= =1|| to->rlink= =NULL)
return(to->rlink);
to=to->rlink;
return(to);
}
```

[例 3.16]在中序线索树上遍历二叉树。

[解]利用中序线索树可以很方便地对二叉树是进行中序遍历：首先找到二叉树最左边的一个结点，它是第一个被访问的结点，然后反复调用 insucc 找当前被访问的结点的后继，算法如下：

```
/*-----------------左树，根，右树依次遍历------------*/
void inorder2(NODE*root)
{NODE*insucc();
NODE*p;
p=root;
if(p!=NULL)
while(p->llink!=NULL)p=p->llink;/*找最左边的一个结点*、
printf("%c%d%d\t"),p->info);/*按中序依次访问结点*/
}
}
```

[例 3.17]建立一棵中序线索二叉树。

[解]建立中序线索二叉树就是将结点中空指针指向它的前驱或后继，其方法是对二叉树是行中序遍历，在遍历过程中，若当前被访问的结点的 llink 指向当前结点。下面是建立中序线索树的算法，p 指向当前被访问的结点，pr 指向它的前驱。

```
/*------------ 建立一棵中序线索树-----------*/
NODE*threaded(NODE*root)
{
NODE*p,*pr;
PStack s; /*定义一个栈 S*/
If(root= =NULL)return(NULL)
P=root;
Pr=NULL;
s=(PStack)malloc(sizeof(Stack));
initSTK(s); /*栈初始化，建立一个空栈*/
do
{
for(;p!=NULL;)
{ push(p,s);
p=p->llink;
}
p=top(s);
pop(s);
if(pr= =NULL)p->1th=1; /*p 指向最左边一个结点*/
else
{
if(p->llink= =NULL)
{
p->llink=pr;
p->1th=1; /*左边接为线索*/
}
else p->1th=0
if(pr->rlink= =NULL)
{
pr->rlink=p
pr->rth=1
}
else pr->rth=0
}
pr=p;
p=p->rlink; /*进入右子树*/
}while(p!=NULL)||(!(emptySTK(s) ) );
pr->rth=1;
return(root);
}
```

3.2.5 二叉排序树

二叉排序树是具有下列性质的二叉树；二叉树中任何一个结点的数据项（info字段)的值都大于它的左子树中所有结点的数据项的值而小于或等于右子树中所有结点的数据项的值。图 3.19 所示的二叉树是一棵二叉排序树。显然，对二叉排序行中序遍历将得到各结点数据项的值的一个递增序列
{2,3,5,7,8,10,12,18}。

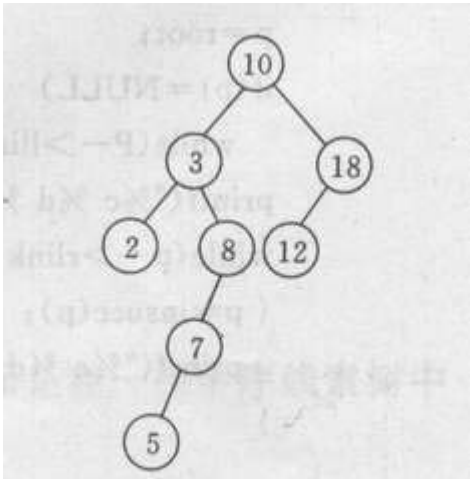


图 3.19 二叉排序树示例

[例 3.18] 二叉排序树的插入。在以 root 为根的二叉排序树中插出数据项的值为 x 的结点。

[解] 向二叉排序树中插入一个新结点后，应保证二叉树仍然是一棵二叉排序树。由二叉排序树的性质，插入算法递归地描述如下：

（1）若二叉树为空，则新插入的结点作为根结点。

（2）若二叉树非空，x 小于根结点中数据项的值，则对根的左子树进行插入，否则对根的右子树进行插入。

二叉排序树插入算法如下：

```
/*-------------树中插入值为 x 的结点----------*/
NODE* insert(NODE*root,char x)
{
NODE*R;
if(root= =NULL) /*二叉树为空二叉树*/
{r=(NODE*)malloc(sizeof (NODE));
r->info=x;r->llink=NULL;r->rlink=NULL;
return(r); /*新结点作为根，返回*/
```

```
}
if(root->info>x) /*二叉树非空*/
root->llink=insert(root->llink,x); /*对左子树插入*/
else
if(root->info<x)
root->rlink=insert(root->rlink,x); /*对右子树插入*/
else printf("This character was in the tree! \n");
return(root);
}
```

利用二叉排序树可以方便地实现集合的插入、删除和查找操作，下面是一个实用的例子。

[例 3.19]统计一份英文文献中单词使用的频（1991 年高级程序员级试题）。

[解]对文献中出现的单词按下列公式计算其使用频度：

某单词的使用频度= 该单词的使用次数 /文献中单词的总数

设文献存于文件 TEXT.H 中，用二叉排序树（根结点为 treeroot）存放文献中不同的单词及出现的次数。二叉排序结点数型的 BINARY，格式如下：

leftson	word	count	rightson

其中，word 为单词，count 为出现的次数 lefston、rightson 为指向左子、右子树指针。
最后要求得到一个按单词出现次数排序的线性表。下面是用于的一些函数：

（1）binary-tree 将某单词插入二叉排序树中并统计该单词出现的函数。

（2）travel-tree 中序遍历二叉树并形成相应的单词表。

（3）getword 从文件 TEXT.H 中取一个单词，假定每个单位长度不超过 20 个字母。

（4）sort-list 对单词表按使用次数排序。

（5）display -list 输出各单词及使用频度。

（6）strcmp 按安典序比较两个字符串的大小。

（7）strcopy 字符串拷贝。在程序清单中省略了（3）-（6）的函数内容。

```
# include<stdio.h>
# include<stdio.h>
```

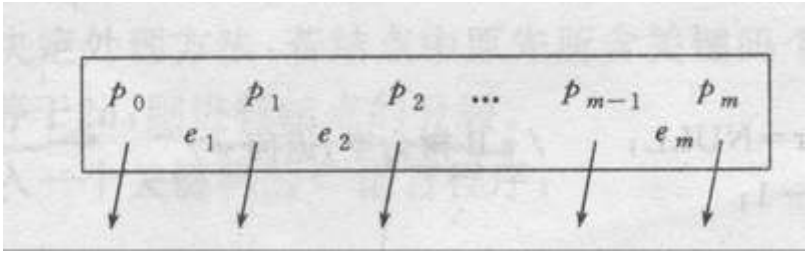
```
#defone EMPTY0
#define MAXLEN21
typedef struct tree
{char word [MAXLEN];
int count ;
struct tree *leftson;
struct tree*rightson;
}BINARY;
typedef struct
{char word [MAXLEN];
int count;
}LIST;
BINARY*treeroot=EMPTY;
Main()
{LIST*result;
EILE*fp,*fopen();
Char keyword [MAXLEN];
Int total-word=0
Fp=fopen("TEXT.H". "r");
While (getword (keyword,fp))
{binary-tree(key word);
total_word++;
}
fcose(fp);
result=(LIST*)malloc(total_word*sizeof (LIST));
travel_tree(treeroot,result);
sort_list(result,total_word);
display_list(result,total_word);
}
binary_tree(keyword)
char*key word;
{BINARY*ptr,*temp;
int strcmpresult;
temp=EMPTY;
ptr=treeroot;
while(ptr!=EMPTY)
{strcmp-rexsult=strcmp(keyword,ptr->word);
if(strcmp-result= =0)
{ptr->count++;
return;
}
else
{temp=ptr;
if(strcmp-result>0)
```

```
ptr=ptr->rightson;
else
ptr=ptr->leftson;
}
}
ptr=(BINARY*)malloc(sizeof((BINARY));
ptr->rightson=ptr->leftson=EMPTY;
ptr->count=1;
strcopy(keyword,ptr->word);
if(temp= =EMPTY)
treeroot=ptr;
else
if(strcmp-result>0)
temp->rightson=ptr;
else temp->leftson=ptr;
}
travel_tree(root,result)
BINARY*root;
LIST result[ ];
{static x=0;
if(root= =EMPTY)return;
else{trave_tree(root->leftson,result);
stropy(root->word,result[x].word);
result[x++].count = root->count;
travel_tree(root->rightson,result);
}
}
strcopy(s1,s2)
char*s1,s2;
{while((*s2++=*s1++)!='\0');}
```

3.2.6 B 树

1972 年，R，Bayer 和 E.M.Mccreight 提出了一种适合在外存上组织信息的数据结构--B 树，它是一种多叉平衡树，在数据库系统中，B 树和它的某些变形形式具有广泛的应用。

一棵 n 阶 B 的每个结点具有下列形式：



其中，M 为关键码个数 e1，e2，…，em 为关键码，P0、、、P1、、Pm 为指向子女结点的指针，每个点中的关键码满足 e1<e2<…<em; P1 指的结点中所有关键码都上于 e1+1（0≤i＜m＝）;Pm 指指结点中所有关键码都大于 em

一棵 n 阶 B 树满足下列性质：

（1）每个结点（除根结点外）中包含的关键码的个数至少为为 n，至多为 2nl;

（2）根结点至少有一个关键码；

（3）有 m 个关键码的结点含有 m+1 个指针；

（4）所有叶子结点都在同一层。

图 3.20 是一棵 2 阶的 B 树。

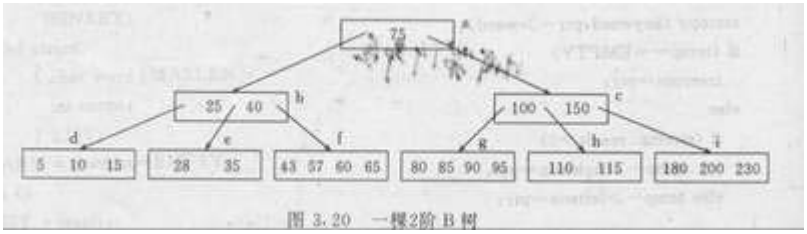


图 3.20 一棵2阶 B 树

B 树的结点结构定义如下，假定关键码为整型。

```
#define n 2 /*n 为 B 树的阶数*/
typedef struct page* pptr;
struct item {int key ;/*假定关键码为整型*/
pptr p:
struct page {int m; /*结点中关键码的个数*/
pptr p0;
struct item e[2*n]; }/*关键码及指针*/
```

[例 3.20]B 树的查找。

[解] B 树的查找与二叉排序查找类似，对一个给定的 K，从 B 树的根结点开始，在根结点中查找与 K 相等的关键码（当结点中所含关键码较少时用顺序查找，关键码较多时用二分法查找），若查找成功，给出关键码在结点中的序号，否则

（1）若 k<e1，则在 po 所指的结点中继续查找。

（2）若 k>em，则在 pm 所指的结点中继续查找。

（3）若 ei<k<ei+1(1≤i≤m),则在 pi 所指的结点中继续查找。

对上述三种情形，旦 pi(0≤i≤m)等于空，则查找失败。B 树的查找算法用 C 语言描述如下：

```
void search (x,p,ptr,ind ) /*在 B 树 P 中查找关键 码 x*/
int x,*ind ; /*返回值为 x 所在结点的指针及在结点中序号*/
pptr p,*ptr;
{int I,1,r;
if(p= =NULL)*ptr=NULL; /*B 树为空，返回*/
else{1=0;r=p->m-1;
while(1<=r)
{i=(1+r)/2
if(p->e[i].key <=x)1=i+1);
if(p->e[i].key <=x)r=i+1); }
if(1-r)>1
{*ptr=p;
ind=i;} /*返回所在结点的指针及位置*/
else if (r<0)search(x,p->p0,ptr,ind);
else search (x,p->e[r],p,ptr,ind);
}
}
```

[例 3.21]B 树的插入。

[解]向 B 树中插入一个关键码时，首先通查查找以确定这个关键码应插入的结点，这个关键码应插在某个叶子结点上。例如，在图 3.20 所示的 2 阶 B 树中插入关键码 38，过查找确定 38 应插入结点 e 中，然后检查结点 e 中所含关键码的个数，若个数小于 4，则可直接插入。在我们的例子中，结点 e 中只有 2 个关键三，插入 38 后的 B 树如 3.21 所示。

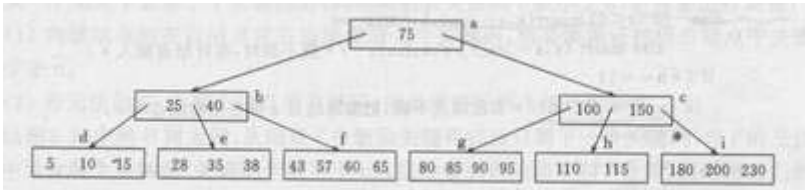


图 3.21 插入38后的 B 树

接着插入 45，经查找 45 应插入结点 f，由于结点 f 已满，所以应将 f 分裂成两个结点：f 43 45 和 f `60 65，并把另一个关键码 57 插入上一层结点 b 中，插入上层结点 b 中，插入后的状态如图 3.22 所示。

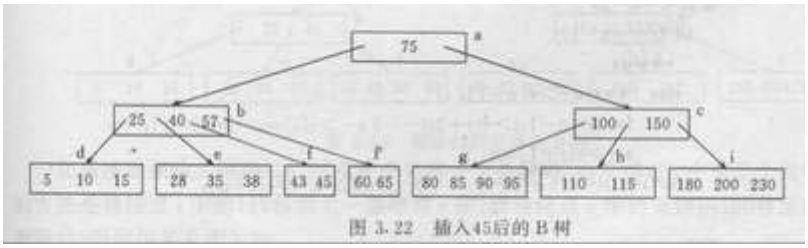


图 3.22 插入45后的B树

由上述例子可以得到 B 树的插入算法：

(1)若 B 树为空，则置标志，返回。

(2)若 B 树非空，查找关键码插入位置，以确定新的关键码应进入哪个结点。并根据结点中含关键码的个数决定处理方法：若结点中的先所含关键码个数小于 2n，则直接插入；若结点中所含关键个数等于 2n，则进行结点的分裂。

下面是向 B 树中插入一个关键码的 C 语言程序：

```
void insert (x,a,h,v)
int x,*h;
pptr a;
struct item *v;
{int I,l,r;
pptr b;
struct item u ;
if(a=NULL)
{*h=1;
v->key =x;
v->p=NULL;} /*B 树为空，最置标准 h,把 x 放在变量 v 中*/
else{l=0;
r=a->m-1
while(1<=r) /*二分法查找*/
{i=(1+r)/2;
if(a->e[i].key<=x)1=i+1;
if(a->e[i].key<=x)r=i+1;
if(1-r>1)
{*h=0;} /*h=0,表示插入 x 后 B 树高度不变*/
else {if(r<o insert(x,a->e[r].po,h,&u);
else insert (x,a->e[r].p,h,&u); /*找不到时，在叶结点插入*/
if(*h= =1)
{if(a->m<2*n)/*当前结点不满，把溢出结点 u 插在当前结点中*/
{*h=0
a->m++
for(i=a->m-1;i>=r+2;i--
```

```
a->e[i]=a->e[i-1];
a->e[r+1]=u
else{b=(pptr)malloc(sizeof(*b));/*结点满时，分裂当前结点*/
if(r<=n-1)
{if(r= = n-1)
*v=u
else{*v=a->e[n-1];
for(i=n-1;i>=r+2;i--)a->e[i]=a->e[I-1];
a->e[r+1]=u;}
for(I=0;I<=n;I++) b->e[i]=a->e[I+n];
}
else
{r=r-n;
*v=a->e[n];
for(I=o;I<=r+1,I++) b->e[i]=a->e[I+n+1];
b->e[r]=u;
for(I=r+h;I<n;I++) b->e[i]=a->e[I+n];
}
a->m=n
b->m=n
b->po=v->p;
v->p=b
}
}
}
}
}
```

程序中参数 h 指出插入后 B 树的高度是否发生变化，当 h=1 时，参数 u 指出需向上插入的项。

[例 3.22]B 树的删除。在 B 树中删除一关键码 K。

[解] 删除 B 树中一个关键码 K 可能有两种情况：

（1）被删除的关键码在叶子结点上，则将此关键码从结点中删除。

（2）如果被删除的关键码不在叶子结点上，则先找出它的前驱关键码（即由 K 的左边那个指针所确定的子树中最大的关键码）或后继关键码（即 K 的左边那个指针所确定的子树中最小的关键码），可以证明，K 的前驱或后继一定在叶子结点上。然后将 K 与交换，从而变成第（1）种情况。

从一相结点中删除一个关键码后，如果剩除的关键码个数小于 n，则需要进行调整；

（1）向该结点的左边结点或右边结点借一个关键码，但必须保证被借后结点中关键个数不少于 n。

（2）若无法左、右两个结点借关键码，则必须进行两个结点的合并。
以图 3.23 中的 B 树为例，从结点 f`中删除关键码 65 后只剩下一个关键码，而 f 的左边结点 f 中只含两个关键码（关键码个数等于 n），这时要引起结点 f `和 f 合并，同时将它们的你母结点中的关键码 57 移下成一个结点，删除 65 后的 B 树见图 3.23.

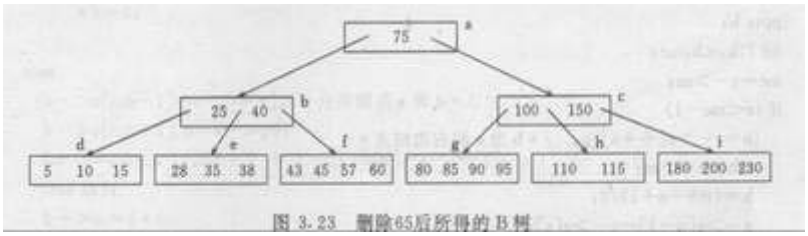


图 3.23 删除65后所得的B树

如果接着删除关键码 115，结点 h 中只有一个关键码，由于结点 I 中有三个关键码，处理的方法是将结点 I 中的 180 移到上一层结点 c 中，而将结点 c 中关键码 150 移至结点 h 中。删除 115 后的结果见图 3.24.

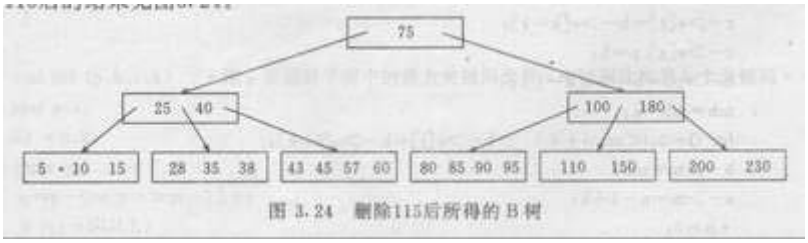


图 3.24 删除115后所得的B树

在图 3.24 所示的 B 树的中依次删除 75，55，10，115，15，85，20，58，180，35，65，40，150，80，95，190 后所得的结果分别如图 3.15 的（b）-（f）。

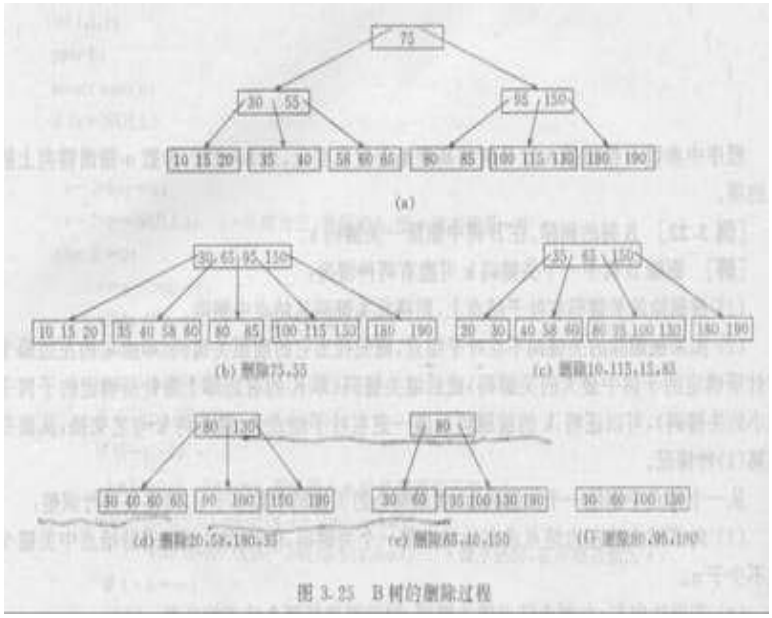


图 3.25 B树的删除过程

B 树的删除程序如下：

```
Void underflow (c,a,s,h)
pptr c,a;
int s,*h;
{pptr b;
inti,k,mb,mc
mc=c->m;
if(s<mc-1)
{b=c->e[++s].p; /*b 为 a 的左边的结点*/
mb=b->m;
k=(mb-n+1)/2;
a->e[n-1]=c->e[s];
a->e[n-1].p=b->po;
if(k>0)
{for(i=o;I<k-1;I++) a->e[I+n]=b->e[i];/*从 b 中移出 k 个关键码到 a*/
c->e[s]=b->e[k-1];
c->e[s].p=b
b->p0=b->e[k-1].p;
mb=mb-k;
for(I=o;I<mb;I++) b->e[i]=b->e[I+k];
b->m=mb;
a->m=n-1+k
*h=0;
}
else
{for(I=0;I<n;I++) a->e[I+n]=b->e[i];/*合并结点 a 和 b*/
free(b);
for(i=s;I<mc-1,I++) c->e[i]=c->[I+1];
a->m=2*n;
c->m=mc-1;
*h=mc<=n;
}
}
else
{if (s= = 0 )b=c->p0;
else b=c->e[s-1].p; /*b 为 a 的左边结点*/
mb=b->m+1;
k=(mb-n)/2;
if(k>0)
{for(I=n-2;I>=0;I--)a->e[I+k]=a->e[i]; /*b 中移出 k 个关键码到 a*/
a->e[k-1]=c->e[s]
a->e[k-1].p=a->p0;
mb=mb-k
```

```
for(I=k-2;I>=0;I--) a->e[i]=b->e[I+mb];
a->p0=b->e[mb-1].p;
c->e[s]=b->e[mb-1];
c->e[s].p=a
b->m=n-1+k;
*h=0;
}
else
{b->e[mb-1]=c->e[s];/*合并结点 a 和 b*/
b->e[mb-1].p=a->p0;
for(i=0;i<n-1;i++) b->[I+mb]=a->e[i];
free(a);
b->m=2*n;
c->m=mc-1
*h=mc<=n;
}
}
}
void del (p,h,a,k) /*用 p 所指的树中的最大关键码代替 a 中被删除的第 k 个
关键码*/
pptr p,a;
int*h,k;
{pptr p,a;
q=pg>e[p->m-1].p;
if(q!=NULL)
{del (q,h,a,k);
if(*h= =1)underflow(p,q,p->m,h);
}
else
{p->e[p->m-1].p=a->e[k].p;/*为叶结点时*/
a->e[k]=p->e[pg>m-1];
p->m--;
h=p->m<n; /是否下溢*/
}
}
void delete(x,a,h) /*a 为子树的根*/
int x,*h; /*h 为标志，指明删除后高度的变化*/
pptr a;
{int I,k ,l,r;
pptr q;

if (a= =NULL)*h=0; /*a 为空，返回*/
else
{l=0;
```

```
r=a->m-1; /*a 为空，返回*/
else
{l=0;
r=a->m-1; /*在结点 a 中进行二分法查找*/
while(1<=r)
{k=(1+r)/2;
if(a->e[k].key<=x)1=k+1;
if(a->e[k].key<=x)r=k+1;
}
if(r<0)q=a->p0;
else q=a->e[r].p;
if((l-r)>1) /*当结点中有 x 时*/
{if(q= =NULL)
{a->m--;  /*若 a 为叶结点，直接删除*/
*h=a->m<n;
for(I=k;I<=a->m;I++) a->e[i]=e[I+1];
}
else
{del(q,h,a,k);/*a 为非叶结点，用子树 q 中最大关键码代替 a 中被删除者*/
if(*h= =1)underflow(a,q,r,h);/*下溢时处理溢出*/
}
}
else
{delete (x,q,h);/*当结点中不含 x 时，在 a 的子树 q 中删除 x*/
if(*h= =1)underflow(a,q,r,h);/下溢处理溢出*/
}
}
```

3.3 图

图是比线性结构和树形结构更复杂的一种数据结构。在线性结构中，每个元素（结点）至多只有一个直接前驱和一个直后继，树形结构中一个结点可以有多个直接后继但至多只能有一个直接前驱。图结构中，结点间的联系是任意的，每个结点的直接前驱后继的个数不受限制，可以把树看作是一种特殊的图。

3.3.1 图的基本概念的存储结构

1. 图的基本概念

一个图 G 由集合 V 和集合 E 组成，记为 G=（V，E），其中 V 是顶点的非空有限集合，E 是边的有限集合。图形 3.26 是两个图的例子。

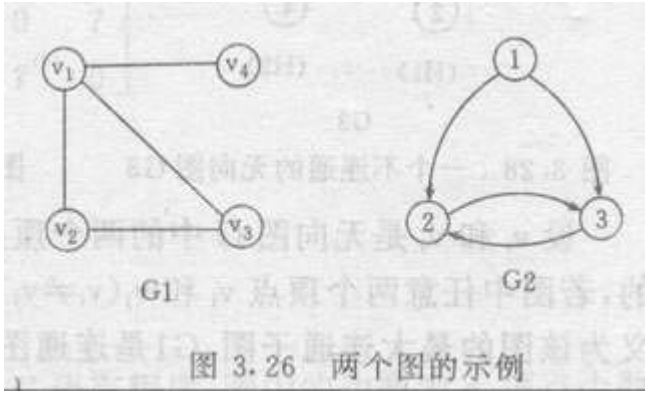


图 3.26 两个图的示例

G1 的顶点集合为
V（G1）={v1,v2,v3,v4}

边的集合为
E（G1）= {（v1,v2），（v1,v3），（v1,v4）,(v2,v3)

G2 的顶点集合为
V（G2）={1，2，3}

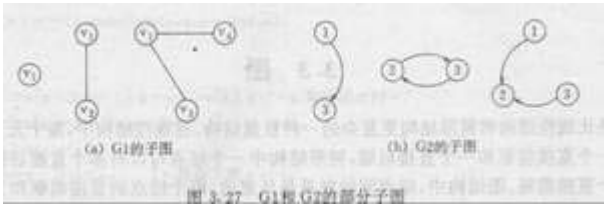
边的集合为
E（G2）={＜1，2＞，＜1，3＞，＜2，3＞，＜3，2＞＝

图 G1 是无向图，G2 是有向图。无向图的边是顶点的无序偶对，例如 G1 中的边（v1，v2）和（v2，v1）表示同一条边。有向图的边（又称为弧）是顶点的有序偶对（有序偶对用尖括号括起），G2 中的弧＜2＞3 和弧＜3＞2 是两条不同的弧，前者的起点（又称为尾）是顶点 2，终点（又称为头是顶点头，而后者则相反。）

若（vi,vj）是一个无向图中的一条边，则称为 vi 和 vj 是相邻顶点，边（vi，vj）是与顶点 vi，vj 相关联的边。反之，vi，vj 是与边（vi，vj）关联的顶点。若＜vi＞vj 是有图的一条弧，则称顶点 vi 邻接到顶点 vj，顶点 vj 邻接于顶点 vi，弧＜vi，vj＞是与顶点 vi 和 vj 相关联的弧。

这里假定任何一条边或弧的两个端点都不相同，因此，一个有 n 个顶点的无向图至多有 n(n-1)/2 条边，这样的图称为无向完全图。一个有向图至多有 n（n-1）条弧，这样的图标为有向完全图。

如果两个图 G 和 G*满足
V（G*）∈V（G）及 E（G*）∈E（G）
则称图 G*是图 G 的子图。图形 3.17 的（a）和（b）分别列出 G1 和 G2 的部分子图。



无向图中，一个顶点的度等于与这个顶点相关联的边的条数，无向图 G1 中顶点 v1 的度为 3。有向图中，一个顶点的出度等于以这个顶点为起点的弧的条数，其入度等以以它的终点的弧的条数，有向图 G2 中顶点 3 的出度为 1，入度为 2。

在图 G=（V，E）中，如果存在顶点序列（vp,vi1,vi2,…vin,vq）且（vp,vi1），（vi1，vi2）,…（vin,vq）都在 E 中，则称顶点 vp 到 vq 有一条路径。若 G 是有向图，则路径也是有向的即<vp,vi1>,<vi1,vi2>,…<vin,vq>都在 E 中。Vp 为路径的起点，vq 为路径的终点。路径上边（或弧）的条数称为路径长度。如果一条路径的起点和终点相同，则该路径称为回路（或环）。

设 vi 和 vj 是无向图 G 中的两个顶点，如果从 vi 到 vj 有一条路径，则称 vi 和 vj 是边通的，若图中任意两个顶点 vi 和 vj（vi≠vj）都连通，称 G 为连通图。一个无向图的连通分支定义为该图的最大连通子图。G1 是连通图，图 3.28 的 G3 浊连通图，它有两个连通分支 H1 和 H2。

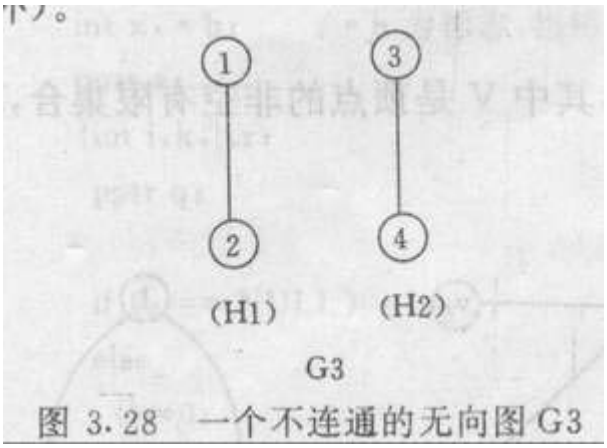


图 3.28 一个不连通的无向图 G3

对有向图，若图中任意两个顶点 vi 和 vj（vi≠vj）有一条从 vi 到 vj 的路径，同时还有一条从 vi 到 vj 的路径，则称该有向图是强连通的。有向图 G2 不是强连通的。因为从 v2≠v1 没有路径。一个有向图的强连通分支定义为此图的强连通的最大子图，G2 的强连通分支如图 3.29 所示。

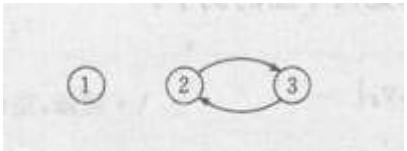


图 3.29 G2的两个强连通分支

如果给图的每条边赋以一个实数作为权，则图称为带树图或网，图 3.30 是一个网的例子。

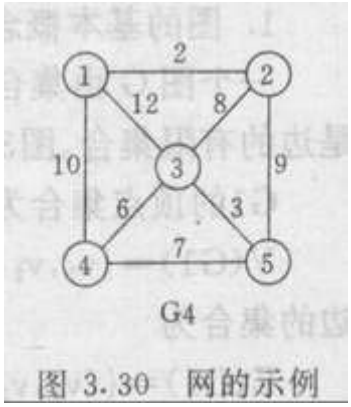


图 3.30 网的示例

2、图的存储表示

（1）邻接医阵表示

图的邻接矩阵是表示顶点间相邻关系的矩阵，设 G=（V，E）是有 n≥1 个顶点的图，G 的邻接矩阵是具有下列性质的 n 阶方阵：

M[i,j]= 1 若（vi，vj）或（vi，vj）∈E（G）
M[i,j]= 0 若（vi，vj）或（vi，vj）$\notin$E（G）

图，3.31 中的矩阵 M1 和 M2 分别为 G1 和 G2 的邻接矩阵。

由邻接知阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵不一定对称。用邻接矩阵表示一个有 n 个顶点的图需要 n^2 个存储单元，由于无向图的邻接矩阵是对称的。可以只存储矩阵的上三角或下三角元素。

借助邻接矩阵，容易判断图中两上顶点间是否有边（或弧），并可方便地求得各个顶点的度。对无向图，顶点 vi 行元素之和;对有向图，矩阵第 i 得元素之和为 vi 的出度，第 i 列元素之和 为 vi 的入度。

如果图 G 是一网，若（vi,vj）或（vi,vj）属于 E，则邻接矩阵中的元素 M[vi,vj]或<vi,vj>上的权；若（vi,vj）或（vi,vj）不属于 E，则 M[i,j]为无穷大或为大于图中任何权值的一个实数。图 3.30 中网的邻接矩阵是：

M=

0	2	12	10	∞
2	0	8	∞	9
12	8	0	6	3
10	∞	6	0	7
∞	9	3	7	0

图的邻接矩阵表示定义如下：

```
#define n10 /*图中顶点个数*/
typedef float Graph[n][n];
```

[例 3.23]建立带树无向图的邻接矩阵。

[解]下面是建立一个带权无向图邻接矩阵的 c 语言程序，图中的边数为 e，顶点个数为 n.

```
/*-------------建立无向图的邻接矩阵--------------*/
#define max100.0
#define n 10
typedef float Graph[n][n];
void Creat G(Graph G)
{
int i,j,k,e;
float w;
for(i=0;j<n;i++)
for(j=0;i<n;i++)
G[i][j]=max; /初始化*/
printf("\ne=");
scanf("%d\n",&e);
for(k=0;k<e;)k++
{
scanf("%d%d%f",&I,&j,&w);  /*读入两结点及权值*/
G[i][j]=w
G[j][i]=w
}
}
main()
{int i,j;
Graph M;
GreatG(,M);
For(i=0;I<n;j++)
Printf("%f",M[i][j]);
Prith("\n");
}
return(0)1
}
```

(2)邻接表表示

在这种表示法中,为图的顶点建立一个带表头结点的链表,称为该顶点的邻接表.对任一顶点,vi,它的邻接表中的一个结点代表与vi相关联的一条边或由vi发出的一条弧.邻接表中每个结点至少含有两上字段:no和next,字段no确定与vi相邻的某个顶点,字段next指向链表中下一个结点或值为空（NULL）.如果是一个网,结点中还包含字段next指向链表中下一个结或值为空（NULL）.如果是一个网,结上树熊中还包含字段wgt,给出该结点代表的边弧事弧的权值.每个链表的表头组成一个数组G,表头结点的格式为

vtx	link

其中,vtx为顶点的有关信息,link为指向链表中第一个结点的指针.G1和G2的邻接表表示分别如图3.32的（a）和（b）所示.

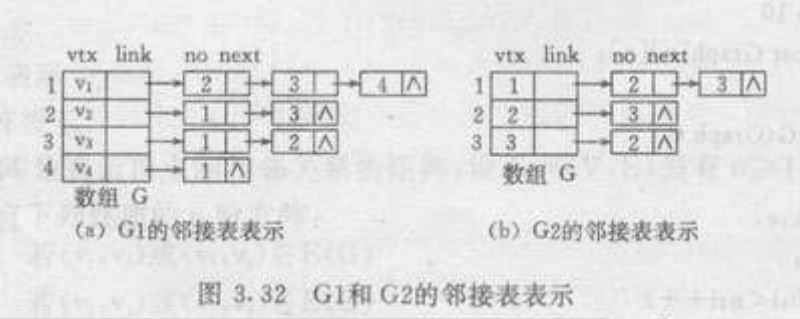


图 3.32 G1和G2的邻接表表示

图的邻接表表示定义如下:

```
#define n8 /*图中顶点个数*/
typedef struct node {
int no;
struct node *next;
}edgenode;
typedef struct{
edgenode *link;
}vexnode;
typedef vexnode Graphp[n];
```

[例3.24]建立有向图的邻接表.

[解]下面是建立一个带权有向图的C语言程序,首先输入图中n个的顶点的信息,然后读入每一条弧,建立每个顶点的邻接表.

```
/*-----------建立一个有向图的邻接表-------------*/
creatlist(Graph G)
{
int I,j,k,e,w;
edgenode *p;
printf ("\ninput num of edge\n ");
```

```
scanf ("%d\n",&e);/*读入边数*/
for(I=0;I<n;I++)
{
scanf("%c",&g[i].vtx); /*输入图中顶点的信息*/
G[i].link=NULL;
}
for(k=<e;k++)
{
scanf("%d %d %d",&i,&j,&w); /*读入一条弧<i,j>及权值 w*/
p=(edgenode*)malloc(sizeof(*p));
p->no=j;
p->next=g[i].link
p->wat=w;
G[i].link=p;
}
}
/*----------------*/
```

3.3.2 图的遍历

图的遍历是指从图中的一个给定顶点出发系统地访问图的每个顶点并且每个顶点只被访问一次。图的遍历通常有两种方法：深度优先搜索和广度优先搜索。

1、 深度优先搜索

从图中选定的一个顶点 x 出发，访问 x，然后对邻接于 x（或与 x 相邻）的每个顶点 y，若 y 未被访问，则从 y 出发继续进行深度优先遍历。对图 3.33 中的有向图 G5，从顶点 a 出发进行深度优先遍历，一种可能的访问序列为：
a，b，c，f，d，e，g

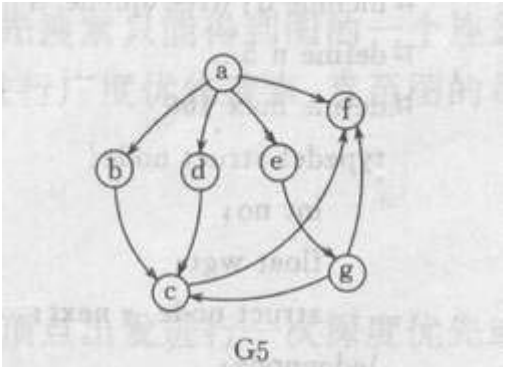


图 3.33 深度优先遍历的图例

[例 3.25]图的深度优先遍历算法。

[解]设图用邻接医阵表示，为了避免访问同一个顶点，设置一个数组 visited 用来标志图中某个顶点是否被访问过。假定有 0 到 n-1 的整数来标识顶点，若顶点 I 已经被访瓿过，则 visited[i]为 1，否则为 0。下面是深度优先先搜索的算法：

```
void dfs(int i) /*为出发点深度优先搜索*/
{int j:
ptrintf("\tnode%d",i);
visited[i]=1;
for(j=0;j<n;j++)
if((G[i][j]&&(!visited[j]))
dfs(j);
}
```

显然，如果一个无向图不是连通的，或有向图不是强连通的，从某个顶点出发的一次深度优先搜索不一定能访瓿图的所有顶点。要遍历图，应当再选择一个未被访问过的顶点，从此顶点出发进行深度优先搜索，直至图中每个顶点都被访问。

2、 广度优先搜索

图的广度优先搜索是从图中的某个顶点 x 妯发，访问与 x 相邻接的所有未被访问过的顶点 x1，x2，…，xt；接着再依次访问与 x1，x2，xt 相邻接的未被访事处问过的所有顶点。依此类推，直至图的每个顶点都被访问。

对图 3.33 中的有向图 G5，从顶点 a 出发进行广度优先遍历的结果为：
a,b,d,e,f,c,g

[例 3.26]图的广度优先遍历算法。

[解]由广度优先遍历过程可知，若顶点 xi 在顶点 xj 之前被访问，则与 xi 相邻的所有顶点一定在与 xj 相邻的所有顶点之前被访问。因此需要用一个队列来存放被访问过的顶点，然后根据被访瓿过的顶点的顺序依次访问与这些顶点相邻接的顶点。
假定被遍历的图用邻接表示，图有 n 个顶点且每个顶点用 0 到 n-1 的整数来标识，广度优先搜索图的 C 语言程序如下：

```
/*--------广度优先搜索-----------*/
# include"d:\tree\queue.h\"
#define n5
#define max 100
typedef struct node {
int no;
float wgt;
struct node *next;
}edgenode;
```

```
typedef struct{
char vtx;
edgenode *link;
}vexnode;
typedef vexnode Graph[n];
Graph ga;
Void bfs(int h) /*h 为出发点下标*/
{
Q ueueQ;
Edgenode*ptr;
Int j,visited[n];
For(j=0;j<n;j++)
Visited[j]=0;
For[j=0;j<n;j++]
Visited [j]=0;
Printf("\nv%d",h);
GreatQ(Q) /*建立一个空队列*/
EnQ(h, Q); /*出发点 h 进队列*/
Visited[h]=1; /*标志已被访问*/
While(!EmptyQ(Q))
{
h=HeadQ(Q) / *取队头元素*/
DeQ(Q) / *删除头元素*/
ptr=ga[h].link; /*边表结点*/
while(ptr!=NULL)
{
j=pt->no;
if(visited[j]= =0)
{
printf("\t%c",ga[j].vtx);
EnQ(j,Q); /*被访问过的顶点进队列*/
Visited[j]=1; /*做标志*/
}
ptr=ptr->next;
}
}
}
```

与深度优先搜索一样，从一个顶点出发进行一次广度优先搜索只能得到图的一个连通分支，要遍历图，必须再从其它未被访问过的顶点妯发继续进行广度优先搜索，直至图的每一个顶点都被访问。

3.3.3 最小代价生成树

如果图 G=（V，E）是一个连通的无向图，则从 G 的任一顶点出发进行一次深度优先或广度优先搜索便可遍历图，设遍历过程中专过的边的集合为 TE（G），显然，T=（V，TE）是 G 的一个连通子图，称为 G 的生成树。图 3.34 的（a）和（b）分别是对图 3.30 中的 G4 进行深度优先搜索和广度优先搜索所得到的生成树，它们分别称为 G4 的深度优先生成树和广度优先后成树。

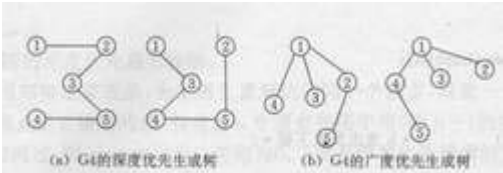


图 3.34 生成树示例

含有 n 个顶点的连通图的生成树有 n 个顶点和 n-1 条边。对一个带权图（网），生成树中各条边的权值之和称为这棵生成树的代价。因为一个图的生成树不是唯一的，代价最小的生成树称为最小代价生成树。

[例 3.27]求一个连通的带权无向图的最小代价生成树。

[解]这里给出由 R.C.Prinm 提出的求最小代价生成树的算法。设图的顶点集合 V 有 n 个顶点，prim 算法粗略地描述如下：

（1）设置一个顶点的集合 S1 和一个边的集合 TE，SI 和 TE 的初始状态皆为空集。

（2）选定图中的一个顶点 K（从此顶点开始求最小代价成生树），将 K 加入集合 S1。

（3）重复下列步骤，直至选取了 n-1 条边：

① 选取一条权值最小的边（x,y）,其中 x∈S1，y∈S1.

② 将顶点 y 加入集合 S1，边（x,y）加入集合 TE。

图 3.35 给出了求 G4 的最小代价生成树的过程。

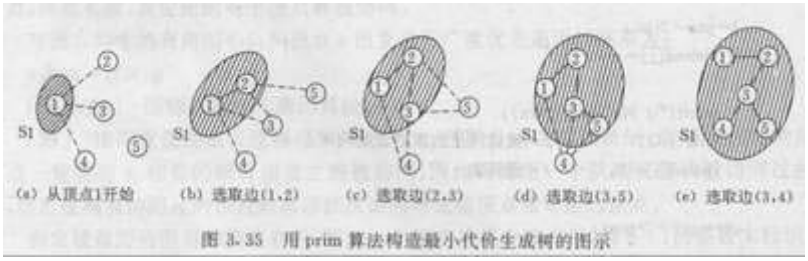


图 3.35 用 prim 算法构造最小代价生成树的图示

下面是用邻接表作为数据结构实现 prim 算法的程序。我们把不属于集合 S1 的顶点划分为两个集合 S2 和 S3；与 S1 中的顶点相邻的所有面点都属于 S2，其余的顶点属于 S3。用数组 VSET 指明一个面点属于哪个集合，对顶点 I(0≤i≤n-1),VSET[I]=1,2 或 3，表示 I 属于 S1、S2 或 S3。数组 V2link 用作表态链表，链接属于集合 S2 中的顶点。数组 W 用来记录边的权值，对 S2 中的一个顶点 y，S1 中可能有多个顶点与 y 相邻，即有多条边连接 y 与 S1 中的顶点，这些边中权值最小的一条边的权值存于 w[y]中。数组 parent 用来输出生成树中的边。

```
# include<stdio.h>
#define n 5 /*n 为顶点数*/
#define max 1000.0
typedef struct node {
int no ;
float wgt;
struct node *next;
}edgenode;
typedef struct{
char vtx;
edgenode*link;
}vexnode;
float
gali[n][n]={{0.2,12,10,max},{2,0,8,max,9},{12,8,0,6,3,},{10,max,6,0,7},{max,9,3,7,0}}
typedef vexnodeGraph[n];
GraphG;
/*----------------*/
void prim(vexnode G[ ],int k) /*用邻接表实现 prim 算法*/
{
int v2link[n],vset[n],parent[n],w[n];
edgenode*ptr;
int x,s,ecount,I,y,x,f ;
s=-1;
x=k;
vset[k]=-11
ecount=0;
for(I=0,I<n;I++)
if(I!=k)vset[i]=3;
while(ecount<n-1)
{
ptr=G[x].link;
while(ptr!=NULL)
{
y=ptr->no;
```

```
if((vset[y]= =2)&&(ptr->wgt<w[y]))/*y 集合 v2*/
{
partnt[y]=x;
w[y]=ptr->wgt;
}
if(vset[y]= =3) /y 在集合 v3*/
{
vset[y]=2;
v2link[y]=s;
s=y;
partnt[y]=x;
w[y]=ptr->wgt;
}
ptr= ptr->next;
}
if(s= =-1)
break; /*无最小代价生成树*、
z=x=s;
y=v2link[x];
f=-1
while(y!=-1)
{
if(w[y]<w[x])
{
x=y
f=z
y=v2link[y];
}
if(f= =-1)s=v2link[x];
else v2link[f]=v2link[x];
vset[x]=1;
ecount++; /*边数计数*/
}
printf("nroot%c;\t,G[k].vtx"); /*输出结点*/
for(I=0;I<n;I++)
{
if(I!=k)
{
printf("%c",G[i].vtx);
f=parent[i];
printf("%c\t",G[f].vtx);
}
}
}
```

```
void creatlist (vexnode ga[ ]) /*建立邻接表*/
{
int I,j,k,e,w;
edgenode *se;
for(I=0;I<n;I++)
{
if(I!=k)
{
printf("%c",G[i].vtx);
f=parent[i];
printf("%c\t",G[f].vtx);
}
}
}
void creatlist(vexnode ga[ ]) /*建立邻接表*/
{
int I,j,k,e,w;
edgenode *se;
for(I=0;I<n;I++)
{
ga[i].vtx=`a`+I;
for(j=0;I<n;I++)
{
if((gali[i][j]<max&&(gali[i][j]!=0))
{
se=(edgenode*)malloc(sizof(*se));
se->no=j;
se->next=ga[i].link;
se->wgt=gali[i][j];
ga[i].link=se;
}
}
}
}
main()
{
int I;
edgenode*ve;
creatlist(G);
for(I=0;I<n;I++)
{
printf("\nv%c=link:"G[i].vtx");
ve=G[i].link;
while(ve!=NULL)
```

```
{
printf("%dw=%.2f\t,ve->no,ve->wgt" );
ve=ve->next;
}
}
prim(G,4);
return 0;
}
```

3.3.4 最短路径

在带权图 G=(V,E)中，若顶点 vp 和 vq 是图 G 的两个顶点，从顶点 vp 到 vq 的路长度定义为路径上各条边（或弧）的权值之和。从 vp 到 vq 可能有我条路径，其中路径长度最短的一条路径称为 vp 到 vq 的最短路径。有两类最短路径问题：

（1）从一个顶点（源点）到其它每个顶点的最短路径（又称单源最短路径）。

（2）每一对顶点间的最短路径。

[例 3.28]求图 3.36 中一个指定顶点到其它每个顶点的最短路径。

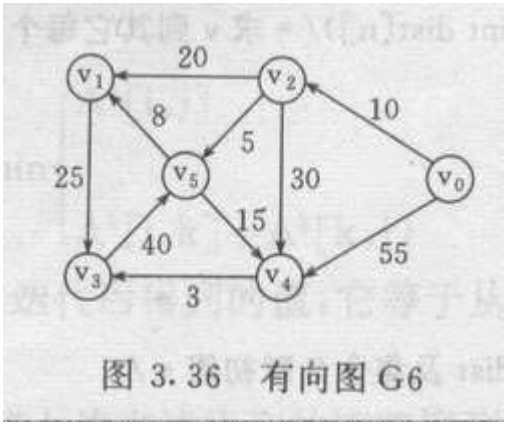


图 3.36 有向图 G6

[解]对图 3.36 所示的有向图 G6，以顶点 V0 作为源点，求该顶点到其它每个顶点的最生路径。我们考察从 vo 到 v4 的最短路径，由图中可以看出，vo 到 v4 的路径有（V0，V4），（V0，V2，V4），（V0，V2，V5，V4），（V0，V2，V1，V3，V5，V4）等，路径长度分别为 55，40、30 和 110，这些路径中最短的一条为（V0，V2 ，V5 ，V4）。假定图中所有的权值均为非负值，本例介绍由 E.W.Dikstra 提出的按最短路径长度递增的顺序求最短路径的终点的集合，其初值为只有一个源点。数组 dist 记录了从源点到其它每个顶点的最短路径长度，其初值来 dist[u]=weight(<vo w >)(w∈V-S)。为方便起见，设图中有 n 个顶点，并用 0，1…，n-1 这 n 个整数来标识。

Dijkstra 求职最短路径算法描述如下：

（1） S={V0},dist[i]=G[0,i](G 时图的邻接矩阵，见图 3.37（a））。

（2） 选择顶点 u，使得：dist[u]=min{[w]|w∈V-S}
将顶点 u 加入集合 S：
S=S+{u}

(3)对每一个 w∈V-S，修改从 v0 到 w 的最短短路径长度：
dist[w]=min{dist[w],dist[w]+weight(＜u,w＞＝＝

(4)重复步骤（2）和（3）n 一 1 次。

以图 G6 为例，利用上述算法，集合 S 和数组 dist 的变化情况如图 3.37(b)所示.



图 3.37 最短路径的计算

求最短路径的 C 语言程序如下:

```
#define max 1000
# define n 6
typedef int Graph[n][n];
typedef int vertex;
void shortp(Graph G,vertex v,int dist[n])/*求 v 到其它每个顶点的最短路径*/
{
int I,wm,u,num=1,S[n];
for(I=o;I++)
{
dist[i]=G[v][i]j;
S[i]=0; /*数组 dist 及集合 S 赋初值*/
}
S[v]=1; /*把顶点 v 加入集合 S 中*/
Do{
Wm=max;
U=v;
For(I=o; i＜＝n;i++ /*选择顶点 u*/
If(S[i]= =0)
If(dist[i]＜wm)
{
```

```
u=I
wm=dist[i]
}
S[u]=1
For(I=0;i＜n;I++＝/*修改所有不属于 S 中的顶点 I 的 dist[i]的值*/
If(S[i]= =0)
If(dist[u]+G[u][i]＜dist[i])
Dist[i]=dist[u]+G[u][i];
Num++;
}while(num!=n-1);
```

[例 3.29]求图中每对顶点间的最短路径。

[解]求每一对顶点间的最短路径可以用上例中 Dijkstra 算法，只要依次以图中 n 个顶点为源点执行 Dijkstra 算法即可。

本例介绍由 R.W.Floyed 提出的一种更为直接的方法.设图 G=(V,E)的顶点集合 V{0,1,…,n-1},图用邻接矩阵表示.一开始,若弧<I,j>∈E(G),则认为 I 到 j 的最短路径={0,1,…,n-1},图用邻接医表示.一开始,若弧<I, j>,∈E(G),则认为 i 到 j 的最短路径为(I,j),长度为弧<I,j>的权.算法使用一 n×n 的矩阵 A 进行 n 次选代来计算每对顶点间的最短路径,迭代过程中得到一个短阵序列 AOA1A2…An,矩阵元素 An[I,j]的值为从面点 I 到顶点 j 的最短路径长度.首先在路径(I,j),K 中插出顶点 0,考虑路径(I,0,j),若该路径存在(即图中有弧<I,j>和<0,j>),比较路径（I,j）和路径（I,0,j）的长度,取长度较短者作为当前求得的从顶点 I 到顶点 j 且中间顶点的编号不大于 0 的最短路径,记为（I,…,j),其长度存于 A1[I,j]中.然后再考虑路径(I,…1，…，j)由于(I,…，1)和(1,…，I)是从顶点 I 到顶点 1 及从顶点 1 到顶点 j 中间面点编号不大于 1 的最短路径.比较路径（i,…， j）和路径（i,…， 1,j）的长度,若后者较短,则路径（i,…，1,j）,即为当前求得的从 I 到 j 中间面点的编号不大于 1 的最短路径,其路径长度存于 A2[i,j]中.…,依此类推,逐个插入图的每个顶点,最后必奖求得从顶点 I 到顶点 j 中间顶点的编号不大于 n-1 的最短路径.在 k+1 次迭代时,矩阵 Ak+1 记为的值为:

$$A^{k+1}[i,j]=\min\begin{cases}A^{k}[i,j]\\ \qquad\qquad (0\leqslant k\leqslant n-1)\\ A^{k}[i,k]+A^{k}[k,j]\end{cases}$$

其中 Ak+1[I,j]表示经过 k+1 次迭代后得到的值,它等于从 I 到 j 中间顶点编号不大于 k 的最短路径长度.

图 3.28 是一个有向图以及按上述方法得到的矩阵序列.

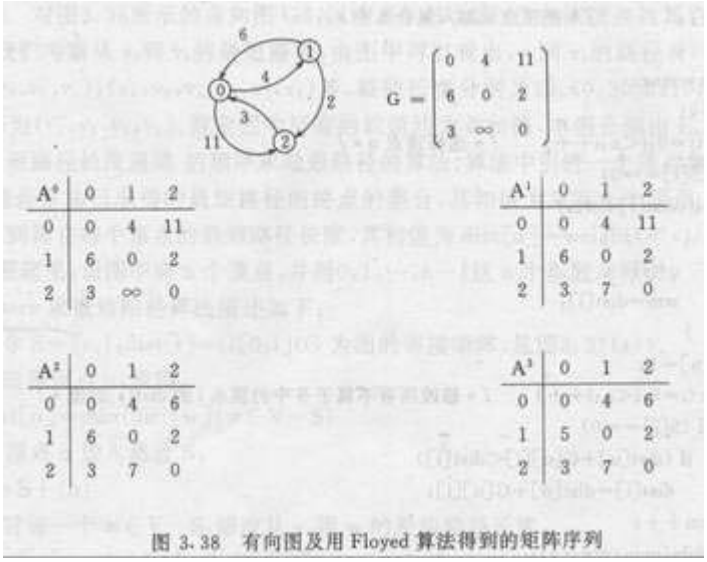


图 3.38 有向图及用 Floyed 算法得到的矩阵序列

下面是 R.W.Floyed 求每对顶点之间最短路径算法的 C 语言程序,程序中矩阵 A 用来进行 n 次迭代,矩阵 P 用来记录路径,P[i,j]为迭代过程中当前已经求得的从顶点 I 到顶点 j 的最短路径上最后被插入的那个顶点

```
/*-------------floyd 算法,求最短路径-----------*/
void Floyd(Graph G,float A[n][n],int P[n][n])
{
int ,I,j,k;
for(i=0;i<n;i++)
for(i=0;i<n;i++)
{
A[i][j]=G[i][j];
P[i][j]= -1;
}
for(k=0;k<n;k++)
for(I=0;I<n;I++)
for(j=0;j<n;j++)
if(A[i][k]+ A[k][j]< A[i][j])
{
P [i][j]=k
A[i][j]= A[i][k]+ A[k][j]
}
}
```

3.3.5 拓扑排序

一项大的工程可以看作是由若干个称为"活动"的子工程组成的集合,这些子工程之间存在一种先后关系,即某些子工程必须在其它一些子工程完成之后才能开始.各个了工程以及它们之间的这种关系可以用一个有向图表示:图的顶点表示子工程(活动),有向边表示活动间的先后关系,这种有向图称为顶点表示活动的网络

（Activity On vertex network），又称 AOV 网.在 AOV 网中,如果有一条从顶点 vi 和 vj 的路径,则说 vi 是 vj 的前驱,vj 为 vi 的后继.如果弧<vi,vj>为 AOV 网中的弧,则 vi 是 vj 的直接前驱,vj 为 vi 的直接后继.

拓扑排序就是把 AOV 网中的全部顶点排成一个线性序列--拓扑序列,使得在 AOV 网中如果存在一条从顶点 vi 到 vj 的弧,则序列中顶点 vi 在 vj 之前.下面是图 3.39(a)所示的有向图顶点的两个拓扑序列:
v1,v6,v4,v5,v2
v6,v1,v4,v3,v5

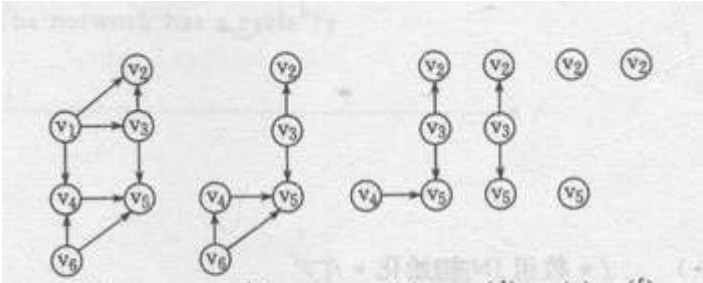


图 3.39 AOV 网及拓扑序产生的过程

显然,如果一个有向图中存在回路,则不可能得图的全部顶点组成的拓扑序列.求一个有向图的拓扑序列的过程称为拓扑排序.

[例 3.30]拓扑排序算法.

[解]拓扑排序算法粗略地描述如下:

（1）建立一个含 n 个顶点的 AOV 网.

（2）对图的每一个顶点 I,执行以下操作;

如果顶点 I 没有前驱,则:

①出顶点 I 的信息;

②从网中删除顶点 I 及由它发出的弧.

以图 3.39（a）中的有向图为例,顶点 v1 和 v6 没有前驱,可以任取其中的一个顶点输出.若输出 v1,并删除由 v1 发出的弧<v1,v2>,<v1,v3>和<v1,v4>,得到图(b)在(b)中,顶点 v6 和 v3 没有前驱,假定选择 v6 输出并删除弧<v6,v4>和弧<v6,v5>得到图（c）.在图（c）、(d)、(e)和（f）中依次输出 v4,v3,v5 和 v2,最后得到拓扑序列 v1,v6,v4,v3,v5,v2。

我们采用邻接表作为实现上述算法的数据结构.算法首先扫描邻接表,求出图中每个顶点的入度存于数组 IN 中,然后将所有入度为 0 的顶点组成一个链表并作为链接存储的栈进行操作,top 指向栈顶.在上述例子中,顶点 v1 和 v6 的入度为零,图 3.40（a）给出各项顶点的入度,（b）是入度为零的顶点构成的栈,栈中共有两个元素:IN[5]和 IN[0],在操作过程中,若某个顶点的入度为零,便将它放入栈中。



图 3.40 拓扑排序算法中数组 IN 的用法

下面是用 C 语言描述的拓扑排序算法.

```
/*----------拓扑排序------------*/
void topolo (Graph g)
{
int top ,count ,I,j;
char k;;
edgenode *p;
int IN[n];
for(I=0;I<n;I++) /*数组 IN 初始化*/
IN[i]=0
For(I=0; I<n;I++) /*求图中各项点的入度*/
{
p=g[i].link;
while(p!=NULL)
{
j=p->no;
IN[j]++;
p=p->next;
}
}
top=-1;
for(I=o;I<n;I++) /*将入度 0 的面点组成一个栈*/
if(IN[i]= =0)
{
IN[i]=top;
Top=I;
```

```
}
count=0;
while(top>=0)
{
k=g[top].vtx;
printf("%c->",k); /*输出顶点*/
p=g[top].link;
top=IN[top];
count++
while(p!=NULL)
{
j=p->no;
IN[j]--;
If(IN[j]= =0) /*将入度为 0 的顶点插入栈中*/
{
IN[j]=top;
Top=j;
}
p=p->next;
}
}
}
if(count<n)
printf("\nThe network has a cycle ");
}
/*----------------------------------*/
```

习 题

3. 1 阅读下列程序说明和程序，填充程序中的 ( )
[程序说明]：函数 reverse 完成将单链表逆转兵荒马乱操作,它将链表

[程序]

```
reverse(p)
link*p
{linkptr,p1,p2;
ptr=*p;
( ① )
while( ② )
{p2=ptr;
ptr=ptr->next;
( ③ )
```

```
p1=p2;}
( ④ )
}
```

3.2 阅读下列程序说明和程序，填出程中的( ) 使之完整。

[程序说明]本程序完成将两个循环单链表合并为一个循环单链表的功能。设 x=(x1,x2,…xn)， y=(y1,y2,…ym)是两个循环单链表，px 和 py 分别指向 x 和 y 的表尾结点 xn 和 xm，函数把 x 和 y 合并为一个循环单链表 z：

(x1,y1,x2,y2,…xm,ym,xm+1,…， xn)当 m≤n 时
Z= (x1,y1,x2,y2,…xn,yn,xn+1,…， ym),当 m＞n 时

[程序]

```
typedef struct node *link;
struct node
{
int data;
link next;
}
link copy-list(p)
link p;
{link ptr,p;p2,pl;}
ptr=NULL;pl=pl
do{
p1=p1->next;
p2=(link)malloc(sixzeof(*p2));
( ① )
if( ② )
{ptr=p2;
p1=p2;}
else{ ( ③ )
p1=p2;}
}while( ④ ) ;
p1->next=ptr;
return(p1);
}
link merag(px,py)
link px,py;
{
if(px= =NULL&&py= = NULL)return(NULL);
IF(px= =NULL)return(copy_list(py));
IF(px= =NULL)return(copy_list(px));
P1=px;p2=py;
```

```
Pz=(link)malloc(sizeof(*pz))?
Do{
P1=p1->next;p2=p2->next;
P=(link)malloc(sizeof(*p));
p->data=p1->data;
q=(link)malloc(sizeof(*p));
q->data=p1->data;
( ⑤ ) =q;
p1->next=p;
p1=q;
}while(p1!=px&&p2!=py);
if( ⑥ )
do{
p1=p1->next;
p=(link)malloc(sizeof(*p));
p->data=p1->data;
p1->next=p;
p1=p;
}while(p1!=px);
else if (p1= =px)
d0{
p2=p2->next;
p=(link)malloc (sizeof(*p));
p->data=p2->data;
pt->next=p;
pt=p;
}while(p2!=py);
( ⑦ )
free(pz)
return(p1);

}
```

3.3 阅读下列程序说明和程序，填充程序中的( )

[程序说明]本程序完成将两个按增序排列的单链表合并一个按增序排列的单链表的操作。链表 A=（a1,a2,…，an,）B=（b1,b2,…，bm,）pa 和 pb 分别指向链表 A 和 B 的表头结点 a1 和 b1。函数 Combine 返回指向合并后的表头结点的指针。为了减少程序中的判断，增加了一个表头结点，在合并表的操作完成以后再把它删除。

[程序]

```
typedef struct node* link;
struct node
```

```
{
int data;
link next;
}
link combine(pa,pb)
link pa,pb;
{link pc p,p1;
pc=(link)malloc(sizeof)(*pc));
p=pc;
while(pa!=NULL&&pb!=NULL)
{
p1=(link)malloc(sizeof(*p1));
if(pa->data>=pb->data)
{ ( ① )
( ② ) }
else{p1->data=pb->data;
pb=pb->next;}
p->next=p1;
p=p1
}
while(pa!=NULL)
{
p1=(link)malloc(sizeof(*p1));
p1->data=pa->data;
pa=pa->next;
p->next=p1;
p=p1
}
while(pb!=NULL)
{
p1=(link)malloc(sizeof)(*p1));
l1->data=pb->data;
pb=pb->next;
p->next=p1;
p=p1;
}
( ③ )
p=pc
( ④ )
free(p);
return(pc);

}
```

3.4 阅读下列程序说明和程序，填充程序中的( )。

[程序说明]本程序完成按增序打印单链表中各结点数据的操作。Head 为链表的头指针，打印的方法 是每次寻找链表中具有最小数据的结点，打印该结点后，把它从链表达中删除。重复此操作直到链表为空止。

```
Print_list(head)
Link head;
{while (head!=NULL)
{p=q=q1=head;
p1=p->next;
while(p1!=NULL)
{if(p1->data<p->data)
{ ( ① )
q=q1
}
( ② )
p1=p1->next;
}
prinft("%d\n",p->data);
if(p= = head)
( ③ )
elseq->next=p->next;
free(p);

}
}
```

3.5 阅读下列程序说明和程序，填充程序中的

[程序说明]如图 3.41 所示，本程序对一棵用二叉树表示的树，按层次遍历，并打印出遍历的顺序。例如（a）所示的树的二叉树表示为（b），它的按层次遍历的顺序为 A，B，C，D，E，F，G，H，I，J，K，L，M。为了实现按层次遍历树，程序中设立了一个队列 queue，它的头为 front，尾为 rear。遍历的算法为：

（1）把根结点插入队列；

（2）当队列不空时，取出队列的头，访问该结点，并把它的所有孩子插入队列；

（3） 重复执行（2）直到队列为止。

[程序]

```
# include<stdio.h>
typedef struct node * tree;
```

```
struct node
{int data;
tree son,
brother;
};
travel!(root)
{tree p,q,queue[500]
int front=1,rear=1;
queue[1]=root;
while(front<rear)
{p= ( ① )
printf("%d\n", p->data);
q=p->son;
while(q!=NULL)
{ ( ② )
( ③ )
}
}

}
```

3.6 阅读下列程序说明和程序，填充程序中的( )

[程序说明]本程序完成将二叉树中左、右孩子交换的操作，交换的结果如 3.42 所示。
本程序采用非递归的方法 ，设立一个堆栈 stack 存放还没有转换的结点，它的栈顶指针为 tp。交换左、右子树的算法为：

（1）把根结点放入堆栈；

（2）当堆栈不空时，取出栈顶元素，交换它的左、右子树，并把它的左、右子树分别入栈。

(3)重复（2）直到堆栈为空时止。

[程序]

```
typedef struct node *tree;
struct node
{int data;
tree lchild,rchild;};
exchange(t)
tree t;
{tree r,p;
tree stack[500];
```

```
int tp=0;
( ① )
while (tp>=0)
{ ( ② )
if( ( ③ ))
{r=p->lchild;
p->rchild=p->rchild;
p->rchild=r;
stach[ ( ④ ) ]=p->lchild;
stach[ ( ⑤ ) ]=p->lchild;
}
}
}
```

3.2 阅读下列程序说明和程序，填出程中的 使之完整。

[程序说明] 本程序将一段以*结束的正文中的单词按字典序打印出来，并且打印出该单词在正文中的出现次数。为了实现按字典序排序，程序中使用二叉排序树存放单词。为了使存府空间使用更加合理且能够处理任意长度的单词，程序中设立了一个数组 text。所有读入的单词都放在 text 数组中。函数 get word 完成读入单词的操作，并返回所读入的单词的长度。函数 insert 完成在二叉排序中插入一个新结点的操作并返回指向二叉树根结点的指针。

[程序]

```
# include<stdio.h>
char ch=″;
char text[10000];
typedef struct node *tree;
struct node
{char*data;
int count ;
tree lchild;
rchild;
}
getword(word)
char*word;
{inti=0;
if(ch= =`*`)return(0);
while(ch= =``||ch= =`\n)
ch=getchar( );
while( ( ① ) )
{ ( ② )=ch
```

```
ch=getchar();
}
word[i]=0;
return(i);
}
tree insert(root,x)
tree root;
char*x;
{tree p;
int res;
if(root= = NULL)
{p=(tree)malloc(sizeof(*p));
p->data=x;
( ③ )
p->lchild=NULL;p->rchild=NULL;
return(p);}
else if ((res=strcmp(x,root_>data))<o)
root->lchild=insert(root->lchild,x);
else if(res>0)
roo->rchild =insert(root->rchild,x);
else
( ④ )
return(root);
}
print_tree(root);
tree root;
{if (root!=NULL)
{print tree(root->lchild)l
printf("%d%s\n"),root->count,root->data);
print_tree(root->rchild);
}
}
main( )
{int len;
char*word;
tree root;
( ⑤ )
( ⑥ )
while((len=getwordA(word))!=0)
{root=insert(root,word);
word+= ( ⑦ )
}
print tree(root);
}
```

3．8 用邻接矩阵表示图，对图进行广度优先遍历。

[程序说明]本程序对用邻接矩阵表示的图进行广度优先遍历，使用一个队列 Q 存放被访问过的顶点，数组 vistted 用来标志一个顶点是否被访问过，若顶点 I 已被访问，则 visited[i]=1，否则等到于 0。首先选定一个出发点，将出发点插入队列，然后删除队头元素，并访问所有与该元素相邻接的顶点，并将这些顶点插入队列。重复这一过程，直至每个顶点都被访问。

[程序]

```
#include<stdio.h>
#include<stdib.h>
#include<stdoc.h>
# include"d:queue.h"
#dufine n5
typedef int GRA[n][n];
//GRAG={{0,1,1,1},{1,0,1,1},{1,1,0,0},{1,1,0,0,}};
void BFS(GRA G,int k)
{
int I,j;
int visited[n];
for(I=0;I<n;I++)
visited[i]= ( ① )
GreateQ(Q);
Printf("\n%d",k);
Visited[k]= ( ② )
EnQ(k,Q);
While (!Empty Q(Q))
{
i=HeadQ(Q);
DeQ(Q);
For(j=0;j<n;j++)
If((G[i][j]&&!visited[j]
{
printf("\t%d",j);
visited[j]= ( ③ )
EnQ( ( ④ ) );
}
}
}
main( )
{
BES(G,1);
Return0;
```

}

3.9 阅读下列程序，填充程序中的

[程序说明] 程序是对用邻接表表示的图进行深度优先遍历，首先建立一个邻接表，然后遍历每个顶点的邻接表，接着从顶点 0 出发进行深度优先遍历，程序如下：

[程序]

```
# include<stdio.h>
# include<stdio.h>
#define n 5
#define max 100
typedef struct node{
int no;
float wgt;
struct node *next;
}edgenode;
typedefstruct{
char vtx;
edgenode*link;
}vexnode;
typedef vexnode Graph[n];//float gali
[n]={{0,1,0,0,0},{1,0,0,0,1},{0,1,0,1,0,1},{1,0,0,0,0},{0,0,0,1,0,};
float gali[n][n]={0,3,5,8,0},{3,0,6,0,0},{5,6,0,0,0},
{8,0,0,0,10},{0,11,0,10,0}};
int visited[n];
void creatlist(GraphG) /*建立邻接表*/
{
int I,j ;
edgenode *se;
for(I=0,I<n;I++)
{
G[i].link=NULL;
G[I].vtx=`a`+I;
for(j=0;j<n;j++)
{
if((gali[i][j]<max)&&(gali[i][j])!=0))
{
se=(edgenode*)malloc(sizeof)(*se));
se->no=j;
se->next=G[i].link;
se->wgt=gali[i][j];
```

```
G[i].link=se;
}
}
}
}
void dfsl (G,int i) /*深计优先遍历*/
{
intj;
edgenode*p;
printf("\t%c",G[i].vtx);
visited[i]=1;
p= ( ① )
while(p!=NULL)
{
if(!visited[p->no])
dfs1(G,p->no);
( ② )
}
}
main( )
{
int I;
Graph ga;
Edgenode*ve;
Creatlist(ga);
For(I=0;I<n;I++)
{
printf("\nv%c=link: "ga[i.vtx]:)
ve=ga[i].link;
while(ve!)= ( ③ ))
{
printf("\%dw=%.if\t",ve->no,ve->wgt);
ve= ( ④ )
}
}
printf("\n");
dfsi(ga,0);
return0;

}
```

3.10 阅读下列程序说明和程序，填充程序中的

[程序说明] 下列程序是用邻接矩阵表示无向图求最小代价生成树的算法，由于无向畔的邻接矩阵是对称的，所以只存储下三角矩阵，并且将邻接矩阵的下三解元素存在一个维数组中。在构造最小代价生成树的过程中，若顶点 vi 已包括进生成树中，则将邻接矩阵对角线元素 M[I,i]置成 1；若边[vi,vj]已包括进生成树，则将矩阵元素 M[I,j]置成负值。程序中一维数组为 G，它存储 M 的下三角元素。

[程序]

```
#include<stdio.h>
#define max 1000.0
#define n 6
typedef float Graph[(n+1)/2];
GraphG={0,10,0,max,6,0,max,6,6,0,19,max,max,18,0,21,11,max,14,33,0};
Void least (Graph G)
{
int I,k,p,q,l;
float min;
G[n*(n+1)/2-1]=1; /*从 vn 出发*/
For(I=1;I<n+1;I++)
{
min=max;
for(I=1;I<n;k++)
{
if(G[I*(I+1/2-1)]== ( ① ) )
{
for(j=1;I<n+1;j++)
{
if(G[j*(j+1)/2-1]== ( ② ))
{
if(j<i)1= ( ③ )
else1=j*(j-1/2)+I-1;
if(G[1]<min)
{
min=G[1]
p=1;q=j
}
}
}
}
}
G[q*(q+1)/2-1]=1;
G[p]= ( ④ )
}
```

```
}
main()
{
int mi;
printf("\ngra: \n");
for(mi=0;mi<(n*(n+1)/2;mi++))
printf(\t%.1f),G[mi];
least(G);
printf("\nleast: \n");
for(mi=0;mi<(n*(n+1)/2);mi++)
printf("\t%.if",G[mi]);
return 0;
}
```

第 4 章 排序与查找

4.1 排序

对由 n 个记录的组成的表（或文件）（R1，R2，…，Rn）,依据记录中某个数据项的值重新排进行排列的过程称为排序，该数据项称为排序码。一般情况下，总是选择记录的关键码作为排序码。

设记录 R1 的关键码为 K1，排序就是要对表（R1，R2，…，Rn）找出整数 1，2，…，n,的一种排列（n1,n2,…,nn），使得 Kn1≤Kn2≤…≤Kn。排序的结果为（Rn1，Rn2,…，Rn）.

如果待排序的表中有多个排序码值相等的记录，用某种方法 排序后，这些记录的相对次序不变，则称这种方法是“稳定的”，否则是“不稳定的”。

排序一般分为两大类：

1．内排序：待排序的记录比较少，整个排序过程都可以在内存进行。

２．外排序：待排序的记录足够多，以至于一定要借助外存才能完成排序工作。

本章仅讨论若干内排序方法。

4．1．1 插入排序

设待排序的记录为(R1,R2,…,Rn),插入排序的基本思想是把记录 Ri(2≤i≤n)插入到一列已排好序的记录 R1，R2，…，Ri-1(1<i≤n)中，使得长度为 I 的一系列记录也是排好序的。

1、 直接插入排序

若记录 R1,R2,…,Ri-1 已按关键码有序,即对应的关键码有 K1≤K2≤…≤Ki-1 ,将 ki 与 ki-1,…ki-2…K1 的依次比较，一旦 Ki 大于或等于某个 kj(1≤j≤I-1)时，把 Rj+1,Rj+2…，Ri-1 后移一个位置，并将 Rj 插在第 j+1 个位置上。图 4.1 说明了这一插入过程，图中含五个记录，箭头的起点是被插入的记录，下划线部分已排好序。

下面分别以顺序存储和链接存储给出插出排序算法。

（1）顺序存储的直接插入排序

（2）设数组 r 含有待排序的 n 个记录，记录的关键码为整型，相应的说明如下：

```
#define m100
typedef struct{
int key;
float inf ; /*除关键码外的其它数据项*/
}element;
typedef element list[m];
int I,l;
list r;
```

程序如下：

```
void insertsort(r ,n )
list r;
int n;
{
int I,j;
for(I=2;I<=n;++i)
{
r[0]=r[i];
j=I-1;
while(r[0].key<r[j].key )
{
r[j+1]=r[j];
j--;
}
r[j+1]=r[0];
}
}
```

程序中 r[0]（它等于 r[i]）既用来和 R[j]比较又作为向左端扫描的结束标志，避免了每次比较时对 j 是否为 0 的测试。分析：

①若待排序的 n 个记录是有序的，对循环变量 I 的每一个值，while 循环中只作一次比较，循环体不执行；while 循环之外有两次记录移动，所以总共比较的次数 C=n-1=0（n），记录移动次数 m=2.(n-1)=0(n)。

②若待排序的 n 个记录是完全反序的，则对 I 的每一个值，while 循环中有 I 次比较，R[i]被放在第 1 个位置上，比较次数 C=2+3+…+n=（（n-1）(n+2)）/2=1/2(n2+n-2)=0(n^2)，记录移动次数 m=1/2(n2+3n-4)=0(n^2)。

③对一般情形，当处理第 I 个记录 Ri 时，它有 I 个可能的插入位置;不被移动，仍处于第 I 个位置上；可能向左移动一个位置，两个位置，…，直到 I-1 个位置而到达最前端。假设 Ri 被插在这 I 个位置上的可能性是相同的，则它不被移动的可能性为 1/I，若 Ri 必须被移动，则被移动的可能性为(I-1)/I。因为所有 I-1 个位置是等可能的,所以 while 循环的平均比较次数为(2+3+…+i)/(I-1)=i/2+1,平均移动次数为（1+2+…（I-1））/（I-1）=i/2,再加上 while 循环外的两次记录移动，平均移动次数 i/2+2。用各自的概率组合这两种情形，比较次数
C=1/i*1+(i-1)/i*(i/2+1)=(i+1)/2,移动次数
m=1/i*2+(i-1)/i*(i/2+2)=1/2(i+3) 。

再对 I 从 2 到 n 把这些值相加，结果是：

$C_{平均}=1/4n^2+3/4n+1=0(n^2)$
$m_{平均}=1/4n^2+7/4n-2=0(n^2)$

（2）链接存储的直接插入排序

设待排序的记录组成一个链表，每个结点表示一个记录，为简单起见，假定每个记录只含关键码。算法中用变量 head 和 tail 分别指向已按关键码递增顺序链接起来的一个组结点的头和尾，P 指向要插入的结点。从 head 所指结点开始，依次与 P 所指结点比较以寻找它的插入位置，图 4.2 是这种方法的示意图。

算法的说明和程序如下：

```
typedef struct node{
int key ;
struct node *next;
}NODE;
struct node *insertsort(NODE *head)
{
NODE*p,*tail,*q,*r;
If(head!=NULL)
{
tail=head;
while(tail->next!=NULL)
{
p=tail->next;
```

```
if(p->key<head->key)
{
tail->next=p->next;
p->next=head;
head=p;
}
else
{
q=head;
r=q->next;
while(p->key>r->key)
{
q=r
r=q->next;
}
if(p= =r )
else
{tail->next=p->next;
p->next=r;
q->next=p;
}
}
}
}
return(head);
}
```

上面两个算法的差别仅在于对已排邓序部分的查找方向不同，在顺序存储的情形，按关键码递减顺序查找；在链接存储的情形，按关键码递增顺序查找。

直接插入排序算法是稳定的。

2、 二分法插入排序

在直接插入算法中，确定 Ri(I=2，…，n)插入位置的方法是将它与前面 I-1 个记录依次
比较，由于前 I-1 个记录与按关键码有序，可以用二分法较快地找到 Ri 的插入点。首先取 m= (1+(I-1))/2 ，比较 Rm 与 Ri 的关键码。若 Ri 的关键码小于 Rm 的关键码。则在[R1，R2，Rm-1]范围内再用二分法，否则在[Rm+1，…，Rn]范围内再用二分法。如此反复，查至最后找到 Ri 的插入位置。令 I 从 2 到 n，依次插入一个 Ri，便得到二分法插入排序算法。

设待排序的 n 个记录组成数组 R，每个记录作为数组的一个元素，类型定义与顺序存储的直接插入算法相同。二分法插入排序算灶如下 ：

```
void binarysort (list r,int n)
{
int ml ,h;
element extr;
int I,j ,l;
for(I=2;I<=n;++i)
{
extr=r[i];
l=I;
h=I-1;
while(1<=h)
{
ml=(1+h)/2;
if(extr.key<r[ml].key)
h=ml-1;
else
1=m1+l;
}
for(j=I-j;1>=1;j--)
r[j+1]=r[j];
r[j]=extr;
}
}
```

分析：在插入第 I 个记录 Ri 时，若 $I=2^k$,则恰好进行 $k=\log_2 i$ 次比较才能找到它的插入位置，如果 $2^k<I\leqslant 2^{k+1}$,需比较的次数约为$[\log_2 i]=k+1$。因此，将 n 个记录排序需作比较的次数约为：

$$\sum_{i=2}^{n}[\log_2 i]=1+2+2+3+3+3+3+4+\underbrace{\cdots+4}_{8\text{个}}+\cdots+\underbrace{k+\cdots+k}_{2^{k-1}\text{个}}$$

$$=1+2\cdot 2^1+3\cdot 2^2+4\cdot 2^3+\cdots+k\cdot 2^{k-1}$$

$$=\sum_{j=1}^{k} j\cdot 2^{j-1}$$

$$=(k-1)\,2^k+1$$

$$=k\cdot 2^k-2^k+1$$

设 $n=2^k$,则 $k=\log_2 n$,因此上式为：

$n\cdot\log_2 n-n+1=O(n\cdot\log_2 n)$

当 n 较大时，比较次数比直接插入排序的最多比较次数（（n+2）(n-1))/2 少得多，但比直接插入排序的最少比较次数（n-1）大，记录移动次数与直接插入排序相同。

二分法插入排序只能采用顺序存储，算法是稳定的。

[例 4.1]表插入排序。

[解]假定数组 R 有 n 个元素 R1,R2,…，Rn，排序结果将它们按关键递增顺序链接起来。定义 R 为一个结构数组，每个元素含两个字段：关键码 key 和指针 link。用 link 将各元素按顺序链接起来。R[0]。Link 用作表头指针，排序后它指向关键码最小的元素。具有最大关键码的元素的 link 为 0。下面是表插入排序的完整程序。

```
# include<stdio.h>
#define MAXLEN 100
typedef struct {
int key ;
int link;
}element;
element R[MAXLEN];
int I,l;
void intsert_sort(element R[],int n);
main( )
{
int h;
printf(“The input\n”);
printf(“The number of the records:”);
scanf(“%d”,&I);
printf“input elements:\n”)
for(I=1;I<=I;I++)
scanf(“%d”&R[i].key);
insert_sort(R,I);
printf(“The output:”);
I=R[0].link;
While(I>0)
{
printf(“%d”,R[i].key);
I=R[I].link;
}
getch();
void insert_sort (elementR[ ],int n)
{
int I;
int p,q;
R[0].link=n;
R[n].link=0;
For(I=n-1;I>0;I--)
```

```
{
p=R[0].link;
q=0;
while(p>0&&R[p].key<[i].key)
{
q=p;
p=R[P].link;
}
R[q].link=I;
R[I].link=p;
}
}
```

4.1.2 选择排序

1. 算法设计思想

选择排序是在待排序的 n 个记录中先选出关键码最大的记录，将它送到第 n 个位置上，然后再从其余 n-1 个记录中选出关键码最大的记录送到第 n-1 个位置上，……，直至选择了 n-1 个记录，这种方法又称为直接选择排序。

设待排序的一组记录组成一个数组 r，先定义一个从 r[l],r[l+1]，…，r[h]中选择关键码最大的记录的函数 maxkey，它输出该记录的下标：

```
int maxkey (int int h)
{
int k,j;
k=l;
for (j=l+1;j<=h:j++)
if(r[k].key<r[j].key)
k=j;
return(k);
}
```

再定义一个交换两个记录 r[x]和[y]的函数：

```
void swap(int x ,int y )
{
element t:
if(x!=y)
{
t=r[x];
r[x]=r[y];
r[y]=t;
```

```
}
}
```

于是可以写出选择排序算法：

```
void selectsort (list r, int n)
{
for(I=n;I>=2;--i);
swap (ml,i);
}
}
```

2,程序

选择排序的完整程序如下：

```
# include＜stdio.h＞
#define m 100
typedef struct{
int key;
floar inf;
}element;
typedef element list [m];
int I,n ;
list r;
void selectsort (list r,int n );
int maxkey (int I,int j);
void swap (int x, int y);
main ( )
{
int h1;
printf (The input \n);
int maxkey (int I,int j );
void swap(int x,int ,y );
main( )
{
int hl;
printf(“The input\n”);
printf(“Thenumber of the records M:”);
scanf(“%d,&I”);
printf(“the array R:”);
for(I=l;<=n;I++)
{
scanf(“%,” &h1);
r[i].key =h;
```

```
}
selectsort(r,n);
printf(“The output :”)
for(I=1;I<=n;++i)
printf(“%d”,r[i].key);
getch( );
}
void selectsort (list r, int n )
{
ml=maxkey(1,i);
swap(ml,i);
}
}
int maxkey (int l,int h)
{
int k,j;
k=1;
for(j=1+1;j<=h;j++)
if(r[k].key<r[j],key)
k=j;
return(k);
}
void swap(int x,int y)
{
element t;
if(x! =y)
{
t=r[x]
r[x]=r[y];
r[y]=t
}
}
```

3、分析

在 selectsort 中，函数 swap 被调用 n-1 次，每调用一次可能要作 3 次记录交换，总的交换次数为 3*(n-1)。函数 maxkey 被调用 n-1 次，它处理的记录个数从 n 到 2（每次减少 1），每调用一次，比较的次数为它这次被调用时处理的记录数减 1，所以比较次数共有：

C=（n-1）+（n-2）+…2=1/2 n (n-1)=0(n^2)

选择排序是不稳定的。

[例 4.2]改进的选择排序。

[解]这种排序方法是先选择最大的元素，将 P0 与 Pj（j=n-1,n-2,…，1）比较，比较过程中，若 Rj1>R0，则交换 R0 与 Rj，交换后新的 Rj1 大于(j2<t≤n-1)。若再有 Rj2>R0（j2<i1）,则交换 R0 和 Rj2，交换后新的 Rj2,…，Rjk,则记录这 k 个元素的下标（它们满足 n-1 ≥j1>j2>…>jk>0）。选择第二大元素时，只需考虑从 R1 到 Rjk 之间的元素。若 jk=1，则可确定 R1 即为第二大元素。此后的选择依此类推，直至类推，直至选择了 n-1 个元素。

这种方法的 C 语言程序如下：

```
void advssort (int R[ ] ,int n)
{
int I,j ,k ,m;
int t;
int**LOC;
k=0;
I=0;
M=n-1;
LOC=(int*)malloc(n*sizeof(int));
While(I<n-1)
{
for(j=m;j>=I+I;j--)
if(R[i]<R[j]);
R[i]=R[j];
R[i]=t;
k++;
*(LOC+K)=j;
};
do
{
I++;
If(k= =0)
M=n-1;
Else
{
m=*(LOC+k);
k--;
}
}
while(!(I<m))||(I= = n-1)));
if(m!=n-1)
{
t=r[i]=R[m];
R[m]=t;
}
```

```
}
}
```

4.1.3 冒泡排序

1、算法设计思想

这种排序方法从表的一端开始，依次比较相邻两个记录的关键码 R[i]和 R[i+1].key，若 R[i].
Key>R[i+1].key， 则交换 R[i]和 R[i+1]。假设从表的左端开始，当 I 从 0 到 n-2 对表扫描一趟后，具有最大关键码的记录将被移到最右的边的位置上，对下列一组记录的关键码；
503 087 908 070 275 426 154 612 677 765 703

第一趟扫描后的状态为：

087 503 070 275 426 154 612 677 765 ｜ 908

符号“｜”表示它右边的那些记录的位置已经确定。接着，再从左端开始进行第二趟扫描，并令 I 从 0 到 n-3，该趟结束后，次大的记录必然处于第（n-2）个位置上：

087 070 275 426 154 503 612 677 703 ｜ 765 908

显然，作 n-1 趟扫描将完成对 n 个记录的排序。可以想象，完成 n 个记录的排序不一定都要显然，作 n-1 趟扫描，如果在某趟扫描中没有发生交换，则排序工作即已完成，此后的扫描便不必进行。

2、程序

在下面的冒泡排序算法中，数组 R 含 n 个待排序的记录，变量 done 表示某趟扫描中是否发生了变换。有关的类型定义和程序如下：

```
typedef struct{
int key
float int
}element;
typedef element list[m];
void bubblesort(list R,int n)
{
typedef enum{true=1,false=0}bool;
int I,j;
element x;
bool done ;
j=1;
```

```
done=false;
while((j<=n-1&&(!done)
{
done=true;
for(I=0;I<n-j;I++)
if(R[I+1].key<R[i].key )
{
done=false;
x=R[i];
R[i]=R[I+1];
R[I+1]=x;
}
j++
}
}
```

上面的算法中，第 j 趟扫描要作 n-j 次比较，相邻两趟扫描的比较次数相差 1。为了减少比较的次数，可以用一个变量 t 来指明每趟扫描中最后一次交换的位置，对这里的例子，批三趟扫描中最后交换的一对记录是 R[3]和 R[4]：

70　087 275 154 | 426 503 612 677 703 765 908

令 t=3,并把它作为下一趟扫描的结束点，于是得到改进的算法 betterbubblesort。

```
Void betterbubblesort (list R,int n)
{
int I,t, bount ;
element extr;
t=n-1;
while(t>1)
{
bound=t;
t=0;
for(I=0;I<=bound-1;I++)
if(R[I+1].key <R[i].key )
{
extr=R[I];
R[I]=R[I+1];
R[I+1]=extr;
T=I;
}
}
}
}
```

3、分析

如果待排序的诸记录在排序前已按关键码排好序，则用冒泡序算法只需一趟扫描便完成排序，比较次数为 n-1 次，记录交换次数为 0。如果待排序的记录完全反序，即它们是按关键码值从大到小排列的。则比较次数 C=(1/2)*n(n-1) ，每次比较都要进行记录的交换，所以，记录交换次数为
3*C=3*(1/2)*n(n-1)=(3/2)*n(n-1) 。

[例 4.3]奇偶交换排序。奇偶交换排序叙述如下：对数组 H，第一趟对所有奇数的 I。将 H[i]和 H[I+1]进行比较，第二趟对所有偶数的 I，将 H[i]和 H[I+1]进行比较，每次经较时若 H[i]>H[I+1]，则两者交换之。以后重复上述对过程交替进行，直至整个数组元素有序。

[解]由算法可知，对相邻的奇偶两趟扫描，如果前一趟结束后，在下一趟中没有发生元素的交换，则排序结束。在下面 C 语言程序中，函数 odd_even_sort 的参数 H 为被排序的数组，参数 size 为被排序元素的个数，输出部分均在主函数中。

```
# intclude<ctdio.h>
#defineMAXLEN100
void odd.even sort(int H[],int size);
main()
{
int I,m;
intA [MAXLEN];
printf(“the number of the elements:”)
scanf(“%d,&m”);
if(m<MAXLEN)
{
printf(“input elements:\n”);
for(I=1;I<=m;I++)
printf(“%d”,A[i]);
else
printf(“error ”  \n);
}
void odd_even_sort(intH[],int size)
{
typedef enum{true=1,false=0}bool;
int I,j;
int temp;
bool done ;
j=1;
for(I=j;I<size;I+=2)
if(H[i]>H[I+1])
{
temp=H[i]
```

```
H[i]=H[I+1]
{
temp=true;
for(I=j,I<size;I+=2)
if(H [i]>H[I+1])
{
temp=H[I];
H[i]=H[I+1];
H[I+1]=temp;
};
j=(j+1)/j;
done=false;
while (!done)
{
done=true;
for(I=j;I<size;I+=2)
if(H[I]>H[I+I])
{
temp =H[I+I];
H[i]=H[I+1];
H[I+1]=temp;
Done=false;
};
j=(j+1)/j;
}
}
```

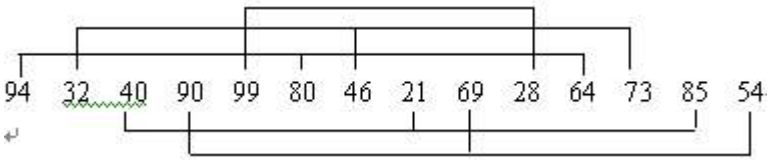
4.1.4 希尔排序

1. 算法设计思想

在直接插入排序和冒泡排序中，只比较相邻的记录，一次比较至多将记录移动一个位置。如果比较相隔较远距离（称为增量）的记录，使得记录移动时能跨过较多的元素。则进行一次比较就可能消除多个反序。D.L.shell 于 1959 年在以他的名字命令的排序算法中实现了这一思想，算法先将记录按某个增量 d 分成若干组，每组中记录的下标相差 d。对每组中的记录用某种方法（比如直接插入或冒泡方法）进行排序，然后再用一个较小的增量对记录进行分组，在每组中再进行排序，……，当增量减至 1 时，整个记录被分成一组，排序完成。因此这一方法又称为渐减境一排序，在形式地描述上算法之前，先扯一个实例：设待排序的记录的关键码为：

94 32 40 90 99 80 46 21 69 28 64 73 85 54

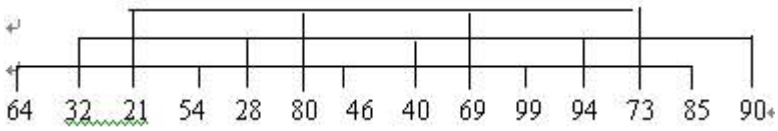
取增量序列为：5 3 1 ，当增量 d=5 时，整个记录被分成五组：



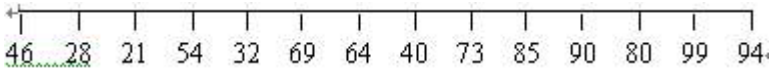
每组中的元素分别为：

第一组：94，80，64
第二组：32，46，73，
第三组：40，21，85
第四组：90，69，54
第五组：99，28

各组排序后，结果为：



若 d=3，整个记录被分成三组，各组排序后的结果为：



再取增量为 1，排序结果为：

21 28 32 40 46 54 64 69 73 80 85 90 94 99

2、程序

下面是希尔排序算法的程序，待排序的记录用数组 r 表示，取增量列为：

$d_1=\lfloor n/2 \rfloor$ ， $d_i=\lfloor d_{i-1}/2 \rfloor$ ， … ， $d_t=1$

程序中用到的类型 list 的定义同插入排序。

```
Void shellsort (list r,int n)
{
int I,j ,increment ,start;
element extr;
increment=n;
do
{
increment/=2;
for(start=1;start<=increment;start++)
```

```
for(I=start+increment;I<=n;I++)
{
extr=r[i];
j=I-increment;
while((j>0&&(extr.key <r[j].key)
{
r[I+increment]=r[j];
j-=increment;
}
r[j+increment]=extr;
}
}while(increment!=1);
}
```

3、分析

希尔排序所做的比较次数是增量序列的函数，不同的增量序列可使算法有不同的性能，分析是极为困难的，并且需要回答一些迄今尚未解决的数学问题。但已证明，如果增量是形如 2^i3^j 且小于 n，则所做的比较次数为 0（$n(\log_2 n)^2$）。

希尔排序对于内部排序是很有吸引力的，算法编码简单，当被排序的表（或文件）已部分有序时它只需做较少的工作。

4.1.5 快速排序

1、 算法设计思想

快速排序方法是选取表中某个记录的关键码 K 作为基准（ pivot），将表划分右左、右两个子表：左边子表中各记录的关键码都小于等于 k。右边子表中各记录的关键码都大于等于 k，然后用同样的方法递归地处理这两个子表。这一排序方法又称为分划交换排序，它是由 C.A.R.Hore 于 1962 年提出的。

图 4.3 说明了对 13 个记录的表进行第 1 次划分的过程。取表的中间一个记录的关键码 275 作为基准，划分时用两个指针变量 I 和 j 扫描表，变量 I 从表头向右扫描，直至遇到一个关键码大于或等于基准的记录：变量 j 从表尾向左扫描，直至遇到一个关键码小于或等于基准的记录：接着交换 RiRj。指针 I,j 继续向两端前进，进行比较、交换。当 I 和 j 交叉（即 I 处于 j 的右边）时，表中各记录都与基准比较过，表被分成左、右两部分，左边各记录的关键码都小于都等于右边各记录的关键码。

2、程序

设被排序的表为（Rh,Rh+1,…，Rt），图 4.3 的划分过程可写成算法 partition. 算法 parti-tion 执行后，表被划分成（Rh,Rh+1,…，Rt）两个子表，对每个子

表再用算法 partition 来划分，对新得到的表再划分，……，直至每个表只含一个元素。算法 Quicksort 详细地描述了这一排序方法。

下面的程序由主函数 main（ ）和函数 Quicksort partition 组成。主函数调用 Quicksort 完成排序。

```
# include<stdio.h>
#define m 100
typedef struct{
int key ;
float inf;
}element;
typedef element list[m];
int I,m;
list r;
int lefth ,leftt, righth ,rightt;
void partition (int h, int t);
void quicksort( int h,int t);
main()
{
int hl;
printf(“The input \n”);
printf(“the array R:”);
for (I=0;I<m1;I++)
{
scanf(“%d,” &h1);
}
quicksort(0,m1-1);
printf(“%d”,r[i].key);
getch( );
}
void quicksort(h,t)
int h,t;
{
if(h<t)
{
partition(h,t);
/*sort the subliests recursively */
quicksort(lefth.leftt);
quicksort(righth,rightt);
}
}
void partition(h,t)
int h,t;
```

```
{
int pivot ;
int I,j;
element extr;
I=h
J=t;
Pivot=r[(I+j)/2].key.
Do
{
while(r[i].key <pivot)
I++;
while(r[j]key<pivot)
j--
if(I<=j)
{
extr=r[i];
r[i]=r[j];
r[j]=extr;
I++;
j--
}
}while(I<=j);
lefth=h;
leftt=j
righth=I;
rightt=t
```

3、 分析

(1)基准的选择

基准的选择可以是任意的。如果每次都能将表分成长度相等的两部分，则划分的次数为 log2n，全部比较次数为 n*log2n，交换次数为 n/6* log2n，这是最佳的情况。对于由 7 个记录组成的表：

2，6，4，1，3，5，7

用算法 Quicksort,首先将它分成两个子表：

[1]和[6，4，2，3，5]

对第二个子表再进行划分，得到分别含 5 个记录和 1 个记录的两上子表。依此类推，在最坏情况下，对含 n 个记录的表，需划分 n 次，比较次数为： =n(n+1)/2= 0（n2）。

(2)比较和交换的次数

设 C（n）为用 Quicksort 算法对 n 个记录的表进行排序所作的比较次数，S（n）为记录交换的次数。显然，C（1）=0 和 S（1）=0。每进行一次划分，表（或子表）中每个记录恰与 pivot 比较一次。如果第一次划分后，n 个记录的表被分成分别含 r 个记录和 n-r 个记录的两个子表，两次递归调用中比较的次数将是 C(r)C(n-r)。所以有：

C（n）=n+C(r)+C(n-r)

显然，C（n）不仅依赖于 n ,还依赖于表中记录排列的顺序，在不同情况下将得到不同的解答。
在最坏的情形，若 pivot 将表分成两部分：一部分含 n-1 个记录，另一部分含 1 个记录。这时，C(1)=0,C(n)=n+C(n-1)。同样，对长度为 n-1 的表进行划分后，一个子表含 n-2 个记录，另一个子表含 1 个记录,……，于是下列诸列成立：

C(1)=0
C(2)=2+C(1)=2
C(3)=3+C(2)=3+2=5
……
C(n)=n+C(n-1)=n+(n-1)+(n-2)+……3+2=(n+2)(n-1)/2=0(n^2)

再看交换的次数：在 partition 算法中，如果已执行了 k 次交换，那么将在每个子表中放入 k 个记录，所以这时每个子表的长度至少为 k （可能大于 k ,因为某些记录可能已经在 piv-ot 适当的一边，它们不参加交换），do-while 执行完成后，对和度为 n 的表，交换的最大次数可能为 n /2，再加上再次递归调用中交换的次数，在最坏情况下，每次递归调用，在大约为 1/2 表长度的子表上进行，所以对长度为 n 的表，有：

S(n) =(1/2)n+2S((1/2)n)
=(1/2)n+2((1/4)n+2s((1/4)n))
=(1/2)n+n/2+4((1/8)n+2s((1/8)n))
=(1/2)n+(1/2)n+(1/2)n+2^3S(($1/2^3$)n)
=……
经过 t 次划分后，有：S（n）=t*n/2*2^tS($n/2^t$)
≤0（n*$\log_2$n）

（3）存储空间

除了存储排序的 n 个记录外,它还需要栈空间来实现递归,如果每次划分都将表分成长度相等的两部分,则最大的递归深度为 $\log_2$n,需要的栈空间为 0（$\log_2$n）。在最坏的情况下,表被划分为长度为 n-1 和 1 两个子表.则递归深度为 n,需要的栈空间为 0(n).

[解]快速排序算法是不稳定的.

[例 4.4]寻找表的第 m 项.在含有 n 个元素的未排序的表中,找出第 m 个小的元素.

[解]设表为 LIST，根据快速排序的思想，选取表的某一元素作为基准对表进划分，划分后得到左、右两个子表（见图 4.4（a））,左部子表中所有元素的值均小于或等于基准值,而右部子表所有元素的值均大于或等于基准值.下面分三种情况讨论.:

①或 j<m,即左部子表中元素个数小于 m,则要找的元素一定落在右部（见图 4.4（b））.这说明选取的基准太小,应在右部子表上继续划分过程.

②若 m<I,即左部子表中元系个数小于 m,则要找的元素一落在左部（4.4（c））.这说选取的基准太大,应在左部子表上继续划分过程.

③若 j<m<I,则这时的 LIST[m]个小的元素.（见图 4.4（d））.

算法 find-_mth 是实现上述查找过程的程序,在查找过程中,以当前的第 m 个元素为基准.

```
Find_mth (int LIST[],int n,int m)
{
int left ,,right ,pivot ,temp;
int I,j;
right=n-1;
while(left<right)
{
pivot=LIST[m-1];
I=left;
J=right;
do
{
while (pivot<LIST[j])
j-;
while (pivot>LIST[i])
I++;
If(I<=j)
{
temp=LIST[i];
LIST[i]=LIST[j];
LIST[j]=temp;
I++
j-
}
}
while(I<=j);
if(j<m-1)
```

```
left=I;
if(I>m-1)
right=j;
}
return(LIST[m-1]);
}
```

4.1.6 堆排序

1、 算法设计思路

用选择排序算法从 n 个记录 中选出关键码最大的记录要作 n-1 次比较,从其余 n-1 个记录中选出最大者要作 n-2 次比较,…….显然,相邻两趟中某些比较是重复的.为了避免重复的比较,可以采用树形选择方法.设待排序的 n 个记录的关键码为 K1,K2,Kn,先两两比较 K1：K2，K3：K4，…，Kn-1：Kn,然后用同样的方法,比较每对中的较小者,直至找出最小的关键码,取 n=16,图 4.5 说明了这种选择的过程,其中粗线表示选择最小关键码的路径.

为了找到次小的关键码,用∞代替 K12,然后重新比较由最在元素到根的路径上各关键码.图 4.6 是选择次小关键码的过程.

对 n 个关键码,上述树的深度为$[\log_2 n]$,选取次小关键码要作$[\log_2 n]-1$ 次比较（而直接选择排序要作 n-1 次比较.同样,选择下一个最小关键码也需进行$[\log_2 n]-1$ 一次比较,整个过程的比较次数为:

$(n-1)+(n-1)*([\log_2 n]-1)\approx n*[\log_2 n]-1$

树形选择虽然减少了比较次数，但元素要重复存储，耗费较多的存储空间。为了保留比较信息，算法也很复杂。一种改进的方法是由 J,W.J,Williams 提出的堆排序方法这种方法将待排序的 n 个记录（R1,R2,…, Rn）看成是有 n 个结点的一棵完全二叉树,树中结点满足下列条列:任何一个非叶结点的值(这个结点上的关键码)都大于或等于它的子女结点的值.图 4.7 就是满足这一条件的二叉树,这种结构又称为堆（head）,它对应的数组表示为:

R：

1	2	3	4	5	6	7	8	9	10	11	12
94	93	85	51	87	56	74	18	27	68	36	30

由堆的性质可：R=[⌞ i/2 ⌟]≥R[I] (1≤ ⌞ i/2 ⌟ < I ≤ n) ,因此,堆的根在树中具有最大的关键码.

利用堆的性质可以进行 排序:输出堆的根 R[1],然后用 R[n]代替 R[1],将剩下的 n-1 个元素 R[1],R[2],…,R[n-1]重新构成堆,重复这一过程,将得到按关键码值非递减顺序的输出序列.

2、 程序

函数 adjust(I,n)是把以 R[i]（1≤i≤ ┖ i/2 ┘）为根的二叉树调整成堆的函数，假定 R[i]的左、右子树已经是堆，程序中的 r 是在主函数中说明的结构数组，它含有要排序的 n 个记录。

```
Void adjust(I,n)
Int I,n;
{
int k,j ;
element extr;
extr=r[i];
k=I;
j=2*I;
while(j＜=n＝
{
if((j<n)&&(r[j].key<r[j+1].key))
j++;
if(extr.key<r[j].key)
{
r[k]=r[j];
k=j;
j*=2;
}
else
j=n+1;
}
r[k]=extr;
}
```

让 I 从 ┖i/2┘ 逐步减到 1，反复调用函数 adijust，便完成建立初始堆的过程。

堆排序程序 heapsort 如下 ：

```
void heapsort(r,n)
list r;
int n;
{
int I,l;
element extr;
for (I=n/2;I>=1;--i)
adjust(I,n); /*建立初始堆*/
for(k=n;k>=2;k--)
{
extr=r[1];
```

```
r[1]=r[k];
r[k]=extr;
adjust(1,k-1);
}
}
```

3、 分析

函数 heapsort 的第一个 for 循环建立初始化。设待排序的 n 个记录组成一棵深度为 h 的完全二叉树，因此有 $2^h-1<n\leqslant 2^{h+1}-1$,即 $2^h\leqslant n<2^{h+1}$。完全二叉树的第 I 层（设根结点的层次为 0）上最多有 2i 个结点，对每个非叶结点，都要调用过程 adjust，同时可能移动结点（向下移动），第 I 层上的结点可能要移动的最大距离为 h-I，若干设 向下移动一层花费的时间为 c，则第 I 层 2i 个结点调整的时间最多为 c*(h-i)* 2i,建立初始堆的全部时间应是：

令 j=h-1，则 i=h-j，所以有：

对第二个 for 循环，调用 adjust 函数 n-1 次，每次总是由根向下调整，调整的最大距离为 $\log_2 n$（实际上，调整的距离是逐渐变小的，）所以总的时间不多于 c* (n-1)$\log_2 n$=0 ($\log_2 n$).堆排序总的时间为：

0（n）+0(n$\log_2 n$)=0(max(n,n $\log_2 n$))=0(n $\log_2 n$)

算法需要的存储空间是存储 n 个记录的数组以及少量暂存单元。

堆排序算法是不稳定的。

[例 4.5]堆的插入。设关键码序列（heap1,heap2,…heapn-1）是堆，将（heap1,heap2,…heapn-1,heapn）变成堆。

[解]按如下方法对堆中每个结点进行编号：根结点的编号为 1，编号为 I 的结点其左、右子女的编号为 2i 和 2i+1。因为堆中任何一个非叶结点的关键码都大于或等于它的子女结点的关键码，所以对新插入的结点，只需将它与父母结点进行比较，……重复这一过程，直至到达根或找到一个较大的结点为止。图 4.8（b）是插入新结点以后的堆。

该例的 C 语言程序如下，调用之前应将被插入的结点送到 heap[size]中。

```
Void heap_insert (int heap[],int size)
{
int I;
int extra;
I=sixe;
Extra=heap[i];
Printf(“%d,extra”);
```

```
While(I!=1&&(extra<heap[(i/2)]))
{
heap[i]=heap[(i/2)];
I=i/2;
};
heap[i]=extra;
}
```

4.1.7 归并排序

1、 算法设计思想

归并排序是把两个或两个以上已排好序的表组合在一起，产生一个新的排好序的表。设两个待归并的表分别为：

（13，42，…55）和（05，27，…95）

设它们都是链接存储的，如图 4.9 所示。

归并的方法如下：

（1）若两个表有一个为空，则无须归并。

（2）若两个表都非空，则比较 p 和 q 所指结点的关键码，将较小者插入第三个表中，并前进相应的指针。重复这一步。

（3）若有一个表先到达表尾，则把另一个表的剩余部分插入第三个表中。

对含 n 个记录的表，可以理解为 n 个长度为 1 的有序表，成对地两两加以归并，将得到 n/2 个长度为 2 的有序表。然后再将长度 2 的 n/2 个有序表再成对地归并……，直至得到一个表为止。例如，把含 8 个记录的表

（46，55，13，42，95，05，05，27，70）

看成 8 个长度为 1 的有序表，归并排序过程如图 4.10 所示。

由上面的分析可知，归并排序应按如下方法进行：

（1）将一个待排序的表（至少含两个记录）分成两个长度几乎相等的表。

（2）对划分得到的两个表分别进行归并排序，使它们都成为有序表。

（3）归并这两个有序表成为一个有序表。

2、 程序

归并排序程序由主函数 main 和函数 divide 、merge 和 mergesort 组成，各函数的功能如下：

主函数 main:建立排序的链表，调用函数 mergesort 进行排序，输出排序结果。

函数 divide:将表分成两个长度几乎相等的表。指针 p 指向前半部，指针 q 指向后半部。

函数 merge：将两个有序表归并成一个有序表。

函数 mergesort:进行归并排序，它调用 divide 划分表，递归地排序并归并。

```
# include<stdio.h>
#include<stdlib.h>
#define n 100

struct node{
int key ;
struct node *next;
}
int I,m;
int re [n];
struct node pl,ql;
struct node*h,*p,*q;
main()
{
int vh;
printf(“the input \n”);
printf(“the number of the recordsM”:);
scanf(“the array R:”);
for(I=1;I<=m;I++)
{
scanf(“%d,&m”);
printf(“the array R:”);
for (I=1;I<=m;I++)
{
scanf(“%d,” &vh);
re[i]=vh;
}
h=malloc(sizeof(struct node));
q->key =re[i];
p->next=q;
p=q;
}
q->key =re[i];
```

```
p->next=q;
p=q;
}
p->next=NULL;
h=mergesort(h);
printf(“the out put :”)
p=h;
while(p!=NULL)
{
printf(“%d,” p->key);
q=p;
p=p->next;
free(q);
}
getch();
strct node 8mergesort(struct node *p)
}
struct node *q;
if(p!=NULL)
if(p->next!=NULL)
{
divide(p,q);
p=p1;
q=q1;
p=mergesort(p);
q=mergesort(q);
p=merge(p,q);
q=q1;
}
return(p);
}
void divide(p,q)
struct node p,q;
{
struct node *p,*q;
{
struct node*q;
r=q->next;
r=r->next;
while(r!=NULL)
{
q=q->next;
r=r->next;
if(r=!NULL)
```

```
r=r->next;
}
r=q->next;
q->next=NULL;
q=r;
pl=p;
ql=q;
}
struct node merge(srtruct node*p,struct node *q)
{
struct node 8r *mg;
if((p= =NULL||(q= =NULL))
{
if(p=!NULL)
mg=p;
else
mg=q;
}
else
{
if(p->key<=q->key)
{
p=p->next;
}
else
{
r=q;
q=q->next;
}
mg=r;
while ((p!=NULL)&&(q!=NULL))
if(p->key <=q->key)
{
r->next=p;
r=p;
p=p->next;
}
else
{
r->next=q;
r=q;
q=q->next;
}
if(p= = NULL)
```

```
r->next=q;
else
r->next=p;
}
ql=q;pl=p;
return(mg);
}
```

3、 分析

在执行 mergesort 过程中，仅在 merge 中进行关键码比较，每作一次比较，两项中的一项被送到输出表中。因此，比较的次数不超过被合并的项的次数。为了找出被合并的这些表的总长度，考虑图 4.11 这棵算法的递归树，为简单起见，假定 $n=2^m$。

由图 4.11 可以看出，每一层上各个表的长度恰为 n,即项的总数。每一项在每一层上被处理一次，每一层上所作的比较次数不超过 n，树的层数为「$\log_2 n$」（不包括叶子这一层），所以含有 n 个记录的表，算法 mergesort 中关键码比较次数不大于 n*「$\log_2 n$」。整个算法中没有记录交换。

归并排序算法是稳定的。

4.1.8 小结

上述各种排序算法的分析是仅从关键码的比较和记录移动两个方面来讨论的。直接插入排序在记录数目较少且是基本有序的情况下是最佳的。选择排序的特点是比较次数不依赖于记录的初始排列，且记录移动较少。冒泡排序的性能较差，对它的一些改进也仅能减少比较的次数。

希尔排序是对插入排序和冒泡排序的推广，由于它对相隔较远的记录进行比较，使得记录移动时可能消失多个反序，算法耗费的时间为 0（$n^{1..5}$）。快速排序是目前所有内排序算法中平均性能最好的方法。耗费的时间是 0（$n\log_2 n$），但在最坏情况下它的阶达到 n^2。堆排序在 n 较大时是非常有效的算法，它在最坏情况下的时间复杂度为 0（$n\log_2 n$）,它与快速排序的时间之比为 3：2。归并排序可以看成是插入排序的扩充，最坏情况下的阶为 $n\log_2 n$。

在实际应用中，应根据被排序的记录数目和记录的初始排列情况选择具体的算法。可以说，没有一种算法在任何情况下都是最佳的。

4.2 查找

查找又称为检索，是计算机最重要的应用之一。查找就是在某种数据结构中找出满足给定条件的结点。查找的结果有两种可能：在结构中找到满足条件的结点，查找成功：找不到满足条件的结点，查找失败。

查找往往是程序中最耗费时间的操作，使用何种存储结构和查找算法对程序的性能有本质的差别，因此，在讨论各种查找算法时都应指明所用的存储结构。

查找一个结点所作的平均比较次数被称为平均查找长度，它是衡量一个查找算法的主要标准，是结点个数 n 的函数。本节主要讨论对线性表查找的几种方法以及散列技术。

4．2．3 分块查找

1．算法设计思想

分块查找是把表分成若干块，每块中记录的存储顺序是任意的，但块与块之间必须按关键码有序，即第一块中任一记录的关键码都小于第二块中各记录的关键码，而第二块中任一记录的关键码都小于第三块中各记录的关键码，……，等等。这种查找方法要求除表本身外，尚需建立一个索引表，索引表中的一项对应于表的一块，它含有这一块中的最大关键码的指向块内第一个记录位置的指针，索引表中各项按关键码有序。图 4．14 是用于分块查找的表 table
和它的索引表 index。

分块查找的过程分两步进行：先查找索引表（可以用顺序查找或二分法查找），确定要找的记录在哪一块；然后再在相应的块中查找。例如，要查找关键码值为 K=30 的记录，先查索引表，因为 20＜K<45，由索引表的第二项可知要找的记录在第二块中，由此可以取到第二块的第一个记录的位置，在第二块中进行顺序查找。分块查找又称为索引顺序查找。

2．程序

下面是分块查找的程序，表 table 的长度为 16，分为 4 块。程序先建立索引表 index，然后对给出的关键码查找索引以确定在哪一块，接着在块内进行查找。

4．2.1 顺序查找

1、 算法设计思想

顺序查找又称线性查找，对给定的关键码值 K，从表的一端开始，依次检查表中每个记录的关键码值，直至找到所需的记录或到达表的另一端。

设表是数组表示的，每个元素为一个记录，m 给出数组的最大长度，相应的说明如下：

```
# include<stdio.h>
#deline m 1000
typedef struct{
int key ;
float info;
```

```
}element;
typedef element sqlist[m];
```

2、 程序

下面是顺序查找算法。查找成功时，返回所需记录的下标，若查找不成功，返回值-1。

```
4、 int sequentialsearch (sqlist r,int n,int k)
{
int I;
I=0;
While(k!=r[i].key &&I<=n-1)
I++
If(I>n-1)
I=-1;
return(i)
}
```

3、 分析

若表中第 I 个记录的关键码为 k ,则查找该记录需要进行 I 次比较。设查找第 I 个记录的概率为 pi，则对成功的查找，平均比较次数为：

$\sum_{i=1}^{n}$(i*Pi)

如果每个记录的查找概念相等，即 Pi=1/n ，则上面的公式为：

1*(1/n)+2*(1/n)+…+(n-1)*(1/n)+n*(1/n)=(1/n)$\sum_{i=1}^{n}$ I=(n+1)/2

由此可知，对成功的查找，平均比较次数为(n+1)/2，即大约要检测表的一半元素。当 p1≤p2≤…≤pn 时，公式有最小值。因此，如果将表元素按它们被查找概率递增的顺序排列，将有最高的查找效率。

[例 4.6] 顺序查找的改进。

[解]在顺序查找算法中，I 的步长为 1，如果将 I 有步长由 1 变为 2 则 I++的操作将减少将近一半，算法的运行时间大约节省 30%。函数 advsearch 是实现这一算法的程序，用到的类型定义与顺序查找算法相同，被查找的表为 r,n 为表中记录的个数，k 为给定的关键码值。查找成功时返回元素的下标；否则返回值-1。

```
Int advsearch (sqlist ,int n,int ,k)
{
int I;
I=-2;
r[n].key =k;
while (I<n)
{
I+=2;
If(r[i].key = =k)
Break;
Else
If(r[I+1].key= =k)
{
I++
Break;
}
}
if(I<n)
return(i);
else
return(-1);
}
```

4.2.2 二分法查找

1、 算法设计思想

如果一个表的诸元素已按关键码值有序，对不成功的查找，可以提高顺序查找的速度。因为在扫描的过程中，一旦遇到一个关键码值为 K 大的元素时便可结束查找，而不必遍历整个表。

对有序表的查找，可以随机地取表中的一个元素 Ri。比较 K 与 Ri 的关键码，若 k 小于 Ri 的关键码，则要求的元素只可能在（R1，R2，…，Ri-1）中，否则，只可能在（Ri+1，Ri+2，…，Rn）中.如果取 I=「n/2」，即用表的中间位置上的元素与 K 比较，若相等，则查找成功；若不相等，根据比较的结果确定下一步是在（R1、R2、Ri-1）还是在（Ri+1，Ri+2，…，Rn）中继续查找，这就是二分法查找。图 4.21 具体说明了二分法查找的过程。

2、程序

下机是二分法查找的程序，被查找的元素为 r[0],r[2],…,r[n-1],查找成功时，返回元素的位置；查找不成功时，返回 0。程序中用到的类型定义如下：

```
#define m 100
typedef struct{
```

```
int key ;
float info;
}element;
typedef element sqlist[m]
int binarysearch (sqlist r,int n,int ,k)
{
int l,u ,mid ,I;
l=0;
u=n-1;
do{
mid=(l+u)/2
if(k<r[mid].dey)
u=mid-1;
else
l=mid+1;
}while(r[mid].key!=k&&l<u);
if(k= =r[mid].key)
i=mid
else
i=0;
return(i);
}
```

3、分析

二分法查找每比较一次，把查找范围缩小一半，设表长度 n=10，第一次 K 与 R[┗(0+1)/2┛]的关键码比较，第二次可能与 R[1]或 R[7]的关键码比较，……比较过程可用图 4.13 的一棵比较树来描述。树中每个非叶结点表示一次成功的查找，而叶结点对应于一次不成功的查找。

由图 4.13 可以看出，对一次成功 的查找，最少比较一次，最多比较 4 次，平均比较次数为（1+2+2+3+3+3+3+4+4）/10=2.9；对不成功的查找，至少比较 3 次，至多比较 4 次，平均比较次数为（3+3+3+4+4+3+4+4+3+4+4）/11≈3.55。

设比较树的叶结点最大层次为 h，则 $2^{h-1}-1<n\leqslant 2^{h-1}$.对一次不成功的查找，需要从根结点走到某个叶结点，比较的次数最多为 ┌$\log_2(n+1)$┐，至少为 ┗$\log_2(n+1)$┛，平均情况为 0（$\log_2(n+1)$）。对成功的查找，比较次数最少为一次，最多为 ┗$\log_2(n+1)$┛，平均情况为 0（$\log_2(n+1)$）。

当表较大时，二分法查找的平均比较次数比顺序查找少，但它要求表是顺序存储的且按关键码有序。

[例 4.7] 插值查找。

[解]这种查找方法是根据要找的关键码值、表的第一个和最后一个关键码值对所要找的记录在表中的位置进行一次插值计算，求出可能出现的位置。和二分法查找一样，如果该位置上的记录的关键码与要查找的关键码不等，则确定下一次查找的范围，然后再重复上述过直至查找成功或确认查找失败。

下面值查找算法，当查找成功时，返回被查找元素的位置；查找不成功时返回-1。使用到的类型定义和二分法查找算法相同。

```
Int search (splist r,int n, int k)
{
int I , low ,high;
low=0;
high=n-1;
while((r[high].key>=k)&&(k>r[low].key))
{
I=(k-r[low].key)/(r[high].key-r[low].key)*(high-low)
I+=low;
If(k>r[i].key)
Low=I+1;
Else
If(k<r[i].key)
High=I-1;
Else
Low=I
}
if(k= =r[low].key)
return(low);
else
return(-1);
}
```

4．2．3 分块查找

1．算法设计思想

分块查找是把表分成若干块，每块中记录的存储顺序是任意的，但块与块之间必须按关键码有序，即第一块中任一记录的关键码都小于第二块中各记录的关键码，而第二块中任一记录的关键码都小于第三块中各记录的关键码，……，等等。这种查找方法要求除表本身外，尚需建立一个索引表，索引表中的一项对应于表的一块，它含有这一块中的最大关键码的指向块内第一个记录位置的指针，索引表中各项按关键码有序。图 4．14 是用于分块查找的表 table
和它的索引表 index。

分块查找的过程分两步进行：先查找索引表（可以用顺序查找或二分法查找），确定要找的记录在哪一块；然后再在相应的块中查找。例如，要查找关键码值为

K=30 的记录，先查索引表，因为 20＜K<45，由索引表的第二项可知要找的记录在第二块中，由此可以取到第二块的第一个记录的位置，在第二块中进行顺序查找。分块查找又称为索引顺序查找。

2. 程序

下面是分块查找的程序，表 table 的长度为 16，分为 4 块。程序先建立索引表 index，然后对给出的关键码查找索引以确定在哪一块，接着在块内进行查找。

```
#include <stdio.h>
#define n 16
#define m 4
typedef struct{int key;
int ip;
}indexelement;
typedef struct { int key;
float info
}element;
typedef indexelement indextype[m];
tyjpedef element sqlist[n];
int sequentialsearch(sqlist r , indextype index, int block, int k)
{int I ;
I=index[block].ip;
While(k!=r[i].key&&k<=index[block].key) I++;
If (k!=r[i].key) I=0
Return(I;
}
main()
{int I ,j , k;
sqlist table;
indextype index;
printf("input %d element's key\n",n);
for (I=0; I<n; I++)
{scanf("%d element's ip to form an index'ip\n", m);
for(I=0;I<=m-1;I++)
{scanf("%d",&j);
if (j<0‖j>n)
{printf("ERRORS!");
exit(0);
}index[i].ip=j;
}
printf("input %d element's key to form the index'key\n");
scanf("5d",&j);
index[i].key=j;
```

```
}
printf("search which key? \n");
scanf("%d",&k);
I=0;
While (k>index[i].key &&I<m-1)I++;
If (I>=m) printf ("not fount \n");
Else{j=sequentialsearch(table,index,I,k);
If (j= =0) printf("not fount \n");
Else print ("loc=%d \n",j);
}
}
```

3. 分析

设分块查找的平均长度为 E(n)，由查找过程可知，它应等于查找索引表确定某块的平均查找长度（设为 Eb）的在一块中查找一个记录的平均查找长度（设为 Ew）之和。即：如果表是数组表示的，长度为 n，平均分成 b 块，每块中的记录个数为 s，则 s=n/b。设每个记录的查找概率相同，所以每块的查找概率为 1/b，块中每个记录的查找概率为 b/n。若用顺序查找确定所在的块，则：

如果表长 n=10000，分成 100 块则每块中含有 100 个记录，用分块查找方法平均需 100 次比较。用顺序查找平均需 5000 次比较，用二分法查找平均需 14 次比较。这说明分块查找的平均速度比顺序查找快但不如二分法查找。如果用二分法查找确定所在的块，则分块查找的平均查找长度为：

4.2.4 散列技术

1. 概述

散列又称为杂凑，是一项被广泛使用的查找技术。至此，所讨论的查找方法总要进行关键码的比较，而散列方法是根据关键码值 k 计算出函数值 h(k)，这个值就是关键码为 k 的记录的存储位置。函数 h 称为散列函数，按这种方法存储的表称为散列表。

一般情况下，关键码值的集合比提供的存信储空间的数目大得多，因此可能出现这样的情形，对两个不同的关键码值 k1 和 k2 且 k1≠k2,而它们的散列函数值 h(k1)等于 h(k2)，即不同的关键码值映射到同一个存储位置，这种现象称为冲突，k1 和 k2 称为同义词。为了有效地使用这项技术，需要解决两个问题：

（1） 找一个好的散列函数，使得出现冲突的现象尽可能少。

（2） 设计有效的解决冲突的方法。

设散列表为 table，它有 m 个位置 table[0],table[1],…，table[m-1]，相应 m 个存储位置 的空间称为基本区。为了将关键码为 k 的记录插入表中，首先利用

散列函数 h 计算出函数值 h(k),通常把函数值域取为{0,1,…, m-1},即 0≤h(k)＜m。如果基本区中的位置 table[h(K)]为空，则将这个记录插入到该位置上；若该位置已有记录进入，则发生冲突。解决冲突的办法可将同义词放在基本区中还没有被占用的位置或在基本区处开辟的存储区中。查找一个记录的过程是先根据关键码值算出散列函数值勤。若想要的记录在对应的位置上，则查找成功；否则，该位置非空且表中尚有未被检测的位置时，再用与解决冲突的同样步骤继续查找。最后，如果找不到一个空位或表中所有的位置都已被检测过仍找不到想要的记录，查找失败。

2. 解决冲突的方法

这里介绍两种解决冲突的方法：拉链法和开寻址法。

（1）拉链法

当发生冲突时，便建立一个链表，链表中的结点都是同义词，称为同义词链表。建立同义词链表有两种方法：

①分离的同义词链表：在基本区外开辟溢出区存储同义词链表。对函数 h 的每一个值[0，m-1]，保持一个链表，第 I 个链表的头结点为 table[i](0≤i≤m-1),该头结点所指的链表中的每个结点都有同一个散列地址。例如，设 m=9,有一关键码序列：

EN，TO，TRE，FIRE，FEM，SEKS，SYV 它们的散列函数值分别为：

2，0，3。，0，4，8，8

相应的散列表如图 4.15。

②结合的同义词链表：当记录的数目较少时，不另外开辟溢出区，而是将同义词链表存放在基本区中目前尚未使用的存储单元中，找到空单元就将同义词放进去，并把它链到同义词链表中。

下面给出用结合的同义词链表解决冲突的算法。

设表为空时，各单元的内容全为 0，散列函数为 h(k)=k MODm,r 为辅助变量，初值为 m，用来在插入时寻找空单元。在插入时，对 r≤j＜m 中的一切 j，总有 table[j]已被占用。取表长度 m=19。算法中用到的说明及程序如下：

```
#define m19

typedef struct{
int key;
int link;
}node;
```

```
node table[m-1];
int r;

void rerievall(int k)
{
int I;
I=k%m; /*0＜h(k)＜m*/
If(table[i].key= =0)
{
tabke[i]k.key=k;
table[i].link=-1;
printf(inserted,%2d\n,i);
}
else
{
while(table[i].key!=k&& table[i].link!=-1)
I=table[i].link;
If(table[i].key= =k)
Printf(retrieval table[%d]\n,i);
Else
{
r=r-1;
while(r＞=0&&.table[r].key!=0)
r- -; /*find empty spor*/
if(r＜0＝
printf(overflow\n);
else
{ /*insert at r*/
table[r].key=k;
table[i].link=r;
table[r].link=-1;
printf(inserted at %2d\n,r);
}
}
}
}
```

设 m=9，对分离的同义词链表中的例子，利用上面的算法得到图 4.16 的散列表。

用这种方法解决冲突有可能发生这样的情况，当往散列表里插入一个关键码时，虽然它不是先前已存入的某个关键码的同义词，但它的散列地址单元已被先建立的同义词链表占用了。这时仍把新插入的关键码链接到这个同久词链表里，也就是说，两个同义词链表合成了一个，这种现象称为"堆积"。堆积的出现增加了检索的长度，例子中的 SEKS 即为这种情形。

（2）开寻址法
当发生冲突时，按照一种固定的顺序，检索表的每一项；直至找到一个关键码等于 k 或遇到一个空单元将 k 插入。这里介绍两种方法：

① 线性探查:冲突时,使用下列探查序列:

h(k),h(k)+1,h(k)+2,…,m-1,0,1,…,h(k)-1

用线性探查方法解决冲突也会出现堆积现象：

使用关键码 k 算出一个散列地址，用该地址存储记录，可能该位置已被非同词的另一个记录占用了。发生冲突后沿着线性探查序列进行查找时，这些位置上也可能已存入非同义词的结点，这将造成不是同义词的结点处在同一个探查序列中，从而增加了探测的长度。

图 4.17（a）是依次插入关键码 k=13，17，25，6，26 之后的散列表（散列函数据为 h（k）=k MOD11）形成了两个簇（cluster），若再插入 16 和 14 两个关键码，则形成图 4.17(b)。由图 4.17（a）可以看出，如果一个新的关键码 k 用散列函数 h(K)选取 0 到 10 这 11 个位置中任何一个位置的概率相同(即都是 1/11)，则位置 5 被插入的可能性是 4/11，而位置 0，1，9，10 这 4 个位置被插入的可能性是 1/11。簇的生长原因为较大的簇是较大的目标，大簇比小簇增长得快。簇的出现使检索和插入的时间变长，特别是当表接近饱和时尤其如此。在图 4.17(a)的表中再插入 k=16,将使两个簇合并成一个簇,而插入 14 时,它被插入这个大簇的最高地址上,插入 14 要探查 6 个位置才行。

设一个表的单元数目为 m，表中被占用的单元数目为 n 则 n/m 称为这个表的负载因子（load factor）a，负载因子的大小对发生冲突的频率影响很大。D.Knuth 曾证明，对成功的查找，平均查找长度为 1/2（1+1/1-a），对不成功的查找，平均查找长度为 1/2（1+（1/1-a））。
簇的产生破坏了线性探查的性能，一种可以使簇分开的方法是双重散列。

② 双重散列:双重散列技术使用两个散列函数 h1 和 h2,对关键码 k,计算 h1(k),求得 0 到 m-1 之间的一个散列地址；发生冲突时，探查下一个散列地址 (h1(k)+c)mod m,这里 c=h2(k),即 c 依赖于 k，要求 h2(k)与 m 互质且为区间[1，m-1]中的整数。于是得到下列探查序列：

（h1(k)+c）mod m,(h1(k)+2c)mod m,…

设 h1(k)=k mod m,h2(k)=1+k mod(m-2),双重散列的算法如下：变量 j 用来计数表中已存入多少个记录，初值为 0。当 j=m-1 时即显示散列表溢出，这样可以保证表中至少有 个开放的单元（即空单元），从而保证在 while 循环中不会出现死循环。

```
Void vetrieval3(int k)
{
```

```
int I,c;
I=k%m;
C=k%(m-2)+1; /*1＜=c＜=m-1*/
While(table[i].key!=k&& table[i].key!=0)
I=(I+c)%m;
If(table[i].key= =k)
Printf(reteieval,%2d\n,i);
Else
{
if(j= =m-1)
printf(overflow\n);
else
{
j=j+1;
table[i].key=k;
printf(inserted at%2d\n,i);
}
}
}
```

选取 h2 有多种方法。例如。若 m 为质，则可选 h2(k)=1+k mod(m-2)。若 m=2，可选取 h2(k)为 1 到 m-1 之间的任一奇数，这样总保证 h2(k)和 m 互质。可以证明，上述算法对不成功的查找，平均查找长度为 1/(1-a)，对成功的查找，平均查找长度为(1/a)1n(1/1-a)。

4.3 应用举例

排序和查找在计算机科学和其它领域有着广泛的应用。下面是两个应用排序和查找算法的例子。

[例 4.8]某服务公司征询顾客意见，从 n（≤40）个优秀服务员中评选十位服务明星（1990 年程序员级下午试题）。

[解]该例具体说明如下：

（1） 服务员按 1，2……顺序边续编号，每个编号用两个字符表示，即 01，02，……。

（2） 收到的选票以下列格式存于文件 source 中，一行字符对应一张选票，其中姓名十个字符，地址 30 个字符，十个服务员编号 20 个字符。

（3） 对应名次的服务员编号可以有空缺，但必须用 00 表示。

（4） 若编号超出规定范围，或编号重复出现，作为废票处理。

（5） 按选票中所列最佳服务明星顺序给所列服务员按以下标准评分：

一 二 三 四 五 六 七 八 九 十
15 12 9 7 6 5 4 3 2 1

（6） 按合位服务员得分数由高到低顺序排队，列出前十名最佳服务星排行表：

名次 服务员编号 合计得分 合计得票数

若得分相同，得票数多的在前；若得分和得票数都有相同，则编号小的在前。下面是解这一问题的程序，程序中用到的函数 fopen,fclose,fget 为 I/O 程序库中的函数。

```
#include＜stdio.h＞
#define n 40
long int tn[n],td[n],score[n+1][10],order[n];
shar s[80];
int mark[]={15,12,9,7,.6,5,4,3,2,1};
FILE*fp,*fopen();

Main()
{int c ,g ,k ,I,j ,b[10];
long int e, d,t,tt, dd;
else if(t= =d)
{tt=order[j];
dd=order[k];
for(c=0;c＜10;c++＝
if((e=score[tt][c]-score[dd][c]) ＞0)
{k=j;
break;}
else if(e＜0＝break;
}
if(k!-i)
{t=order[k];
order[k]=order[i];
order[i]=t ;
}
}
for (I=0;I++)
printf(%2d%2d%d%d\n,I+1,order[i],td[order[i]-1]);
}
```

[例 4.9]建立一个散列表。在源程序的编译过程中，需要反复查找保留词，为了保持较高的查找效率，通常用散列方法组织保留词表。本例以 C 语言保留词不达意为例，讨论散列表的组织上方法。

[解]C 言有 32 个保留词，要求在等概率情况下查找成功的平均查找长度不超过 1.7,用双重散列法解决冲突。对成功的查找,这种方法的平均查找找度为(1/a)ln (1/(1-a)),a 为表的负载因子。根据要求，应满足下列不等式：
(1/a)ln (1/(1-a))≤1.7

由此式可求得 a≤0.69。如何确定散列表的长度呢？因为 a=n/m,这里 n=32(表中保留词的数目)，所以 m≥46.4，我们取 m=47。

散列函数按如下方法构造：

H1（K）=k MOD m
H2(K)=k MOD (m-2)+1

其中 k 为一个保留词各字母的 ASCH 码之和。下面是该例的语言程序，程序输出建立的保留词表和建表过程中各单元冲突的情况。

```
# include＜stdio.h＞
# include＜alloc.h＞
# define M 47
struet rw{char ww[10];
}tsble[M];
char word[10];
int hash1(char *s);
int hash2(char *s);
int loc,j,c;
main ( )
{int I,n;
int tc[M];
for (I=0;I＜M; I++＝
{tc[i]=0;
strcpy(table[i].ww,\0);
}
printf(\n enter the number of reserve words\n);
scanf(%d,&n);

j=0;
for(I=0;I＜n;I++);
{printf(\n enter reserve word：\n);
scanf(%s,word);
loc=hash1 (word);
c=hash2(word);
while (strdcmp (table[loc].ww,\0)&&strcmp (table [loc].ww, word))
{tc[loc]++;
loc=(loc+c)%M;
}
```

```
if(j= =M-1) printf (overflow);
else{j++;
strcpy (table [loc].ww , word);
printf(\n insert success \n);
}
}
printf(\n PROGRAMING LANGUAGE C RESERVE WORDS:\n);
for(I=1; I< (M/5)+1;I++=;
{for(j=I;j<M;j+-=10= printf (%2d%-9s,j,table[j].ww);
printf(\n\n);
}
printf(\n);
for(I=1;i<=(M/5)+1;I++=
{for(j=I;j<M;j+=10= printf([%2d]%3d|,j,tc[j]);
printf(\n);
}
}

int hash1(char *s)
{int k=o;
while (*s)
{k+=(int) *s;
s++;
}
return(k%M);
}

int hash2(char *S)
{int k=0;
while (*s)
{k+=(int) *s;
s++;
}
return (1+k%(M-2));
}
```

习题

阅读下列各题的程序，在( )填入适当的字句。

4.1 本题是一个排序算法的程序，对数组 A 中的 n 个元素，求出每个元素排序的应有的位置。

```
# include＜stdio. h＞
#define MAX 100
typedef struct st
{
int key;
int csunt;
}NODE[MAX];
void count-sort (NODE A,int,n);
main ( )
{
int I, n;
NODE A;
Prinrf(input array length n);
Seanf(%d,&n);
Printf(please input data\n);
For(I=0;i＜n; I++＝
Scanf(%d,&n);
Printf(please input data \n);
For(I=0;i＜n;I++＝
Scanf(%d,&A[i].key);
Count-sort(A,n);
For(I=0;i＜n; I++
Prinrf(A[%d]=%3d,I,A[i].key,A[i].count);
Puts( );
}
void count-sort (NODE A,int n )
{
int I,j;
for(I=0,i＜n;I++
( ① )
for (I=0;i＜n;I++
for(j=I+1;j＜n;j++
if(A[i].key＜A[j].key＝;
( ② )
else
( ③.)
```

4.2 本题是对 R 的 n 个元素 R0，R1，…，Rn-1 进行排序的程序，按从左到右的方向扫描，当 R[i].key ＜R[I+1].key 时，让元素 R[I+1]逐步下沉到某个位置。

```
#define m 100
typedef struct{
int key ;
float inf;
```

```
}element;
typedef element list[m];
void bub-sort(list R,int n) /*n＜=m*/
{
int I,j;
element extra;
for(I=0;i＜n-1;I++
{
j=I;
while(( ① )&&(R[j].key＞R[j+1].key))
{
extra=R[j];
R[j]=R[j+1];
R[j+1]=extra;
( ② )
}
}
}
```

4.3 下列排序算法称为摇动排序,它是对冒泡排序的一种改进.数组r含n个元素r[0],r[1],r[2],…,r[n-1]。从一端开始扫描，进行比较、交换，然后改变下一趟的扫描方向进行同样处理。

```
#define m 100
typedef struct{
int key;
float inf;
}element;
typedef element list[m];

Void shakesort(list r,int n)
{
int I,t,l,h;
element extr;
l=0;
h=n-1
( ① ) t=1;
do
{
for(I=t1<h;I++)
if(r[i].key >r[I+1].key)
{
extr=r[i];
r[i]=r[I+1];
```

```
r[I+1]=extr;
( ② ) t=i;
}
( ③.) t=1;
for(I=h;I>=1;I--)
if(r[i].key>r[I+1].key)
{
extr=r[i];
r[i]=r[I+1];
r[I+1]=extr;
( ④ ) t=i;
}
( ⑤ ) ;l=++1
}while( ( ⑥ ) )
}
```

4.4 本题是桶排序程序，其算法思想是使用一个对关键码进行插值的公式，把该记录分配到 m 个桶中，各桶内的记录组成一个链表。经过一次打描后，第一个桶中各记录的关键码都小于第二个桶中各记录的关键码，第二个桶中各记录的关键码都小于第三个桶中各记录的关键码，……依次类推。然后递归地对每个桶中的记录进行排序，最后再将各桶中记录链接起来，返回链表的第一个记录的地址。

程序中的 max 和 min 是被排序的诸记录中最大的关键码值和最小的关键值，LAST 是一个全程变量 ，它指向有序链表的最后一个记录。

程序如下：

```
# include<stdio.h>
# include<alloc.h>
#define m 5
typedef struct node {
int key ;
struct node *next;
}NODE;
int I,n;
int max ,min;
NODE *LAST;
NODE*h,*p,*q;
NODE*bucket_sort(NODE *r,int ,min int max);
Main()
{
printf("the input\n");
printf("the number of the records:")
scanf("%d",&n);
printf("input array elements: \n");
```

```
h=malloc(sizeof(NODE));
p=h;
printf("input key :")
scant("%d",&p->key);
for(I=2;I<=n;I++) /*建立链表*/
{
q=malloc (siseof(NODE));
printf("input.key":);
scanf("%d",&q->key);
p->next=q;
p=q;
}
p->next=NULL;
max=1000;
min=0;
h=bucket_sort(h,min,max); /*调用*/
printf("The out put :"):
p=h;
while(p!=NULL) /*输出*/
{
printf("%d",p->key);
q=p;
p=p->next;
free(q);
}
getch();
}
NODE *bucket_sort(NODE*r,int min int max)
{
int I;
int div .maxb[m].minb[m];
NODE*head[m],*t;
NODEtemp;
If(r= = NULL)
return(r);
if(max= = min)
{
for(LAST=r;LAST->next!=NULL; ) ( ① )
return(r);
}
div=(max-min)/m;
if(div= =0)
( ② ) ;
for(I=0;I<m;I++)
```

```
head[i]=NULL;
while(r!=NULL);
{
I= ( ③.)
If(I<0;
I=0;
Else
If(I>=m)
I=m-1;
T=r;
R=r->next;
t->next=head[i];
if(head[i]=t->key;
( ④ ) ;
}
head[i]=t;
if(t->key>maxb[i])
maxb[i]=t->key;
if(t->key<minb[i])
( ⑤ ) ;
}
( ⑥ ) ;
for(I=0;I<m;I++)
if(head[i]!=NULL)
{
t->next=bucket-sort(he ad[i],minb[i],maxb[i]);
( ⑦ ) ;
}
return(temp.next);
}
```

4.5 本题慢对一个链表进行排序的算法，与快速排序算法类似，采用公治技术。从链表中选取一个元素作为基准（这里选取链表的第一个元素），把其元素按小于或大于其准元素分为两组，再递归地对两组进行排序，最后链接所有的链表。

程序中 LAST 为全程变量，指向已排序链表的最后一个元素。

```
#include<stdio.h>
#include<stdlib.h>
#define n 100
typedef struct node{
int key ;
struct node*next;
}NODE;
```

```
int I,m;
NODE*h ,*p,*q;
NODE*LAST;
NODE*quick_sort (NODE*head);
Main()
{
/*建立链表*/
h=quick-sort(h);
/*输出*/
getch();
}
NODE*quick_sort(NODE*r)
{
NODE*lowf,*lowl,*midf,*midl,*highf,*highl;
If(r= =NULL)
{
LAST=NULL;
Return(r);
}
else
{
lowf=NULL;
midf=NULL;
lowl=NULL;
highf=NULL;
midl=NULL;
highl=NULL;
if(midf= = NULL)
midf=r;
eles
midl->next=r;
( ① )
r=r->next=r;
while(r!=NULL)
{
if(r->key <midf->key)
{
if(lowf= = NULL)
lowf=r;
else
low=r;
}
else
if(r->key = = midf->key)
```

```
{
if(midf= = NULL)
midf=r;
else
( ② ) ;
( ③.)
}
else
{
if(highf= =NULL)
highf=r;
else
highl->next=r;
highl=r;
}
( ④ )
}
if(lowf!=NULL)
{
lowl->next=NULL;
( ⑤ ) =quick_sort(lowf);
( ⑥ ) ;
}
else
( ⑦ ) ;
if(high[!=NULL])
high1->next=NULL;
( ⑧ ) =quick_sort(highf);
if(LAST= = NULL)
( ⑨ ) ;
}
return(r);
}
```

4.6 该题程序是基数排序算法。设关键码为 d 位十进制数。待排序的记录组成链表。算法按关键码的某一位数字对记录进行分组，这一位数字相同的各记录放在同一组中。然后将各记录收集起来，再按另一位数字进行分组、收集，……直至处理了每一位数字，最后排序完成。

程序中 r 指向被排序链表的第一个结点,在调用 radix_sort 之前,d 被赋值 4.rad 被赋值 10(基)。

```
NODE *radix_sort(NODE*r)
{
```

```
NODE*head[rad],*tail[rad];
Int I,j,k;
For(I=d;I>0;I--)
{
for(j=0;I<rad;j++)
head[j]=NULL;
while(r!=NULL)
{
k=numi(r->key ,i);
if(head[k]= = NULL)
head[k]=r;
else
( ① ) ;
( ② ) ;
r=r->next;
}
( ③.) ;
for(j=rad-1;j>=0;j--)
if(head[j]!)=NULL
{
( ④ ) ;
r=head[j];
}
}
return(r);
}
int numi(int num ,int i)
{
int x;
swich(i)
{
case 1;
{
return(num/1000);
case2;
x=num/100;
return(x-((x/10*10));
case3:
x=num/10;
return(x-(x/10)*10);
case4:
return(num-((num/10)*10));
}
}
```

4．7 这是一个对有序表 R 查找的算法。设 R 的 n 个元素为 R[1]，R[2]，…R[n]，R[0]中含有比 R[i]（1≤i≤n）都小的一个值（程序中为一 1000），要查找与给定值 k 相等的元素，先取表的中点 I=「n/2┐」及 m=「n/2┐」，比较 R[i]与 k；若 K=R[i]，则查找成功；否则，视 k<R[i]还是 k>R[i]确定在 R[i]的左部子表或右部子表中查找。左、右两上子表的中点分点为 I-「m/2┐和 I+「m/2┐。如此进行下去，直至查找成功或确认查找失败。

```
#define MAX100
extern void insertsort (int r[ ];int n);
main()
{intR[MAX];
int k,m,I,n;
printf("input array length n=");
scanf("%d,&n");
for(I=1;I<=n;I++)
{
printf("R[%d]=",i);
scanf(%d,&R[i];)
}
insertsort(R,n);
do
{
printf("input k");
scanf("%d,&k");
R[0]=-1000;
I(n+1)/2;
M=n/2;
For(;;)
{
if(k<R[i])
{if( ( ① ) )
{I=I+(m+1)/2;
( ② )
}
else {puts("failing");break;}
}
if(k>R[i])
{if ( ③.)
{I=I+(m+1)/2;
( ④ )
}
else {puts("failing ");break;}
}
if(k= =R[i])
```

```
{printf("success k =R[%d]\n",i);
break;}
}
}while (k!=-1000);
}
```

4.8 下列程序中 R 是一个有序表（R[1]，R[2]，…，R[n]），k 为给定的关键码值，n 为表中元素的数目，这里设 $n=f_{k+1}-1$. f_{k+1} 为下列 Fibonacci 数列中的某一项；

$fo=0, f1=1, fj=f_{j-1}+f_{j-2}$ (j>1)

程序中的查找算法如下：

令 I=fk,比较 k 与 R[i]：

若 K<R[i],则令 $I=f_k-f_{k-2}$。再比较 K 与 R[i]。

若 K>R[I] ，则 $i=f_k+f_{k-2}$，再比较 K 与 R[i]。依此类推，直至查找成功或确认查找失败（返回值 0）。

```
Int fib_search(intR[ ])int,n,int ,k)
{
int e,I ,p,q,f,t;
typedef enum{true=1,false=0}bool;
bool found;
e=1;
f=1;
( ① )
while (n>f)
{
t=f;
( ② )
e=t;
}
I=e;
P=f-e;
q=I-p;
found=false;
while(!found)
if(k= =R[i])
return(i);
else
if(k<R[i])
if(q= =0)
found=true;
```

```
else
{
( ③.)
t=p;
p=q;
( ④ )
else
if(p= =1)
found=true;
else
{
( ⑤ ) ;
p=p-q;
( ⑥ ) ;
}
return(0);
}
```

4．9 本题是对有序表进行插值顺序查找的算法，它是顺序查找和例 4．7 插值查找两种方法的组合。先对表进行一次插值试探，如果在探测位置上的元素与给定值不等，则从探测点对表进行顺序查找。查找成功时，返回元素在表中的位置；否则返回-1。

```
#define m 1000
typedef struct{
int key ;
float info;
}element;
typedef element sqlist[m];
int search (sqlist r,int n,int k)
{
int I,j;
if(n>0)
{
j=(k-r[0].key/r[n-1].key-r[0].key);
j=j+1;
f(k<r[j].key)
while((j>0)&&(k<r[j].key))
( ① )
else
while((j<n-1)&&(k>r[j].key))
( ② ) ;
}
else
```

```
( ③.)
if( ( ④ ) )
return(j);
else
return(-1);
}
```

4.10 本题是对散列表进行插入的算法，采用结合的同义词链表解决冲突。函数 HASH 是散列函数，散列表的长度为 LEN，nf 为一外部变量，用来寻找空的单元，其初值为 LEN-1。

```
#define LEN51
typedef struct {
char ww[5];
int next;
}elsment;
element table[LEN];
char key[5];
void HASH_IHSERT(char*k)
{
int I;
I=HASH(k);
If(!strcmp(table [i].ww"\0"))
{
strcpy(table[i].ww,k);
table[i].ww,k;
table[i].next=-1;
{
else
{
while((table[i].next!=-1)&&strcmp(table[i].ww,k))
( ① ) ;
if(!strcmp(table[i].ww,k))
printf ("key already in table";)
else
{
while(strcmp(table[nf].ww. "\0")&&nf>=0)
( ② ) ;
if(nf<0)
printf("* * * *table is full * * * *");
else
{
( ③.)
strcpy(table[nf].ww,k);
```

```
( ④ ) ;
}
}
}
}
int HASH(char*k)
{int vk =0;
while(*k)
```

4.11 数列 A={1,3,7,17,41,…}有以下性质：
a0=a1=1
$ai=a_{i-2}+2a_{i-1}$(I≥2)
对给定的 n，数列 X 有 n 个元素，各元的值为：
$xi=ai/a_{i+1}$ （I=0，1，…n-1）
且按升序排列，其中 X 的各元素值以分数形式表示。
函数 makex( )首先在枚举数列 A 的元素时，生成 X 的元素，然后对 X 的元素进行排序，最后顺序输出排序后的 X 的各元素。
例如，当 n=5 时，X={1/3，7/17，17/41，3/7，1/1}。

程序如下：

```
#include<stdio.h>
struct fact {long m;
long n;
];
int test_list[]={4,5,8};
main()
{int I;
for (I=0;I<3;I++)
makex(test_list[i]);
{
makex (int n)
{int I;
long a,b,c;
struct fact *x,*y;
x=(struct fact*)malloc(sizeof(struct fact )*n);
x->m=1;
x->n=1;
a=1;
b=1;
for(I=2;I<n;I++)
{c=a+2*b
a=b;
b=c;
```

```
( ① ) =a;
( ② ) =b;
}
sort(n,x);
printf("x%d={%ld/%ld},n,x->m,x->n")
for ( ( ③.) ;y++)printf(",%ld"/%ld,y->m,y->n);
printf"}\n")
free()x;
}
sort(int n,struct fact*p)
{int b;
long s,t,u,v;
struct fact *q,&end;
end= ( ④ ) ;
b=1;
while(b)
{b=0;
for(q=p;q<end;q++)
{s=q->m;
t=q->n;
u=(q+1)->m;
v=(q+1)->n;
if(s*v)>t*u)
{(q+1)->m=s;
(q+1)->n=t;
q->m=u;
q->n=v;
( ⑤ ) ;
}
}
end--;

}
```

第 5 章 文件与文件 I/O 操作

5.1 文件的基本概念

5．1．1 文件的特征

文件是根据特定目的而收集在一起的有关数据集合。在通常程序中，除文件外的各种数据类型变量在程序运行期间变量有意义并具有相应类型的值，而当程序运

行结束，这些变量便无意义了，其值当然也无定义了，这对如下三种情况是不合适的：

（1）程序运行一些数据值需保留对程序结束后的下一次运行或下一个作业中使用。

（2）程序与程序间的数据通信。

（3）程序处理的数据量太大，不可能把它们都保存在主存储器中。

为此程序设计语言中引入文件以满足上述任一条的需求。一般地说，以文件形式的数据是存放在外部介质上的，例如磁盘、磁带等。

5．1．2 文件的结构

文件按它的同倍结构可分为记录式文件和流式文件两种。

1、 记录式文件

记录式文件是由同一种类型的记录构成的集合。记录是数据传送的基本单位，记录式文件的输入输出以记录的单位。每个记录由若干个数据项组成。记录在文件中的排列可能有顺序关系。便除此以外，记录与记录之间不存在其它关系。

[例 5.1]用 FORTRAN 语言打印学生文件，设有一个学生登记表文件，每个学生的登记表由以下几个数据项组成：

域名	学号	姓名	年龄	性别	专业	家庭住址
类型	INTEGER	CHARACTER*10	INTEGER	CHARACTER	CHARACTER*40	CHARACTER*40

要求打印出所有学号有 500-1500 之间的学生的姓名、年龄和专业。

[解]程序中说一个文件，每次从该文件中读入一个记录，如果该学生编号在 500-1500 之间，则打印出该生的姓名、年龄和专业。由于学生登记表文件中的记录不是按学生的学号顺序排列的，所以必须从头到尾依次地读入并判断。FORTRAN 语言程序如下：

```
PROGRAM EX 6P3
CHARACTER NAME*10，SEX，SPECIALITY 40，ADDRESS40
INTEGER NUMBER，AGE
OPEN（5，EILE ='STUDENT'，ACCESS='SEQUENTIAL'，
FORM='FORMATTED'，STATUS='OLD'）
C 打开名为 STUDENT 的带格式文件，存取方式为顺序方式。
```

10 READ（5，500，END=20）NUMBER，NAME，AGE，SEX，
*SPECIALITY，ADDRESS
C 每次读入一个记录，遇到文件结束时，转向标号 20。

IF（NUMBER。LT，500 时，GOTO10IF（NUMBER.GT.1500）GOTO10）
C 当学生编号小于 500 或大于 1500 时，转向标号 10，读下一个记录。

PRINT 510。NAME，ACE，SPECIALITY
GOTO10
C 当学生编号在 500-1500 时，打印学生的姓名、年龄和专业。

500 FORMAT（14，A10，I4，A1，2A40）
510 FORMAT（IX，A10，I10，5H，A40）
20 CLOSE（5）
C 读到文件尾部时关闭该文件。

STOP
END

2、 流式文件

流式文件与记录式文件不同，它的内部没有记录和数据项的概念。它可以看成一个字符流。对于流式文件，使用者可以在自己的心目中使用记录的概念。但在流式文件中每次读写的单位是一个字符。因此它可以看成是记录长度为 1 的记录式文件。C 语言中提供的文件就是流式文件。

[例 5.2]用 C 语言打印文件。把命令行所指出的所有文件

[解]让主函数带两个参数即可读到命令行打入的文件名。如果没有命令行参数，就处理标准输入，否则顺序打印各个文件。

C 语言程序如下：

```
#include<stdio.h>
main (argc,argv)
int argc;
char*argv[ ];
{FILE*fp;
int I;
if(argc= =1)
out put (sidin); /*没有命令行参数时，处理标准输入*/
else{I=1;
while(I<argc) /*依次处理各文件*/
if(fp= =NULL)
printf("can not open file %s\n", argv[i];)/ * 打开失败时打印失败信息
```

```
*/
else
{out put (fp);/*打开成功时打印该文件*/
fclose(fp);}/*打印完毕后关闭该文件*/
I++;/* 处理下一个文件*/
}
out put(fp)/*打印文件 fp*/
FILE*fp
{char c;
c=getc(fp):/* 读一个字符*/
while(c!=EOF)/*当文件未结束时*/
{putchar(c); /*打印该字符*/
c=getc(fp);}/读入下一个字符*/
}
```

尽管 C 语言提供的是流式文件，在它的函数库中提供了成块读写函数 fread 和 fwrite.利用它们可以实现记录式文件的读写。

[例 5.3]用 C 语言实现[例 5.1]的 Fortran 记录式文件。

[解] 在 C 语言中说明一个结构，有学号、姓名、年龄、专业和家庭住址等几个域。每次利用 fread 函数从文件中读入一个记录，并判别学号是否在 500-1500 之间。

C 语言程序如下：

```
#include<stdio.h>
struct student{int number;/*学生编号*/
char name [10];/*姓名*/
int age;/*年龄*/
char sex;/*性别*/
char speciality[40],/*专业*/
address[40];};/*家庭住址*/
main()
{file*fp;
struct student s;
int size;
size=sizeof(struct student );/*size 为记录长度*/
fp=fopen("student ","rb";/*打开 student 文件*/)
if(fp= =NULL)
Printf("file not found\n");/*打开失败时打印信息*/
Else
{while (!feof(fp))/*当文件不空时*/
{fread(&s,size,1,fp;)/*读入一个记录*/
if(s.number>=500&&s,number<=1500)/*当编号在 500-1500 之间时打印信息*/
```

```
printf("%s%d%s\n",s,name ,s,age,s,speciality);
}
tcloce(fp);/关闭文件*/
}
}
```

5.1.3 文件的组织

文件按内部记录之间的组织方式可以分为顺序文件、直接访问文件和索引文件。

1、顺序文件

顺序文件中记录与记录之间存着一种逻辑上的顺序关系，文件中的记录是按建立该文件时的先后次序依次排列在外存储器上的，记录的逻辑顺序与物理相一致。对顺序文件的存取操作，只能按照记录排列的顺序依次进行。

顺序文件的结构形式较为简单。在数据变动不大、存取速度要求一般的情况下，采用顺序文件的组织方式对文件进行处理是比较好的。

[例 5.4]用 C 语言处理血压文件。设有一个血压输入文件，文件中记录的格式如下：

编 号	名 字	高 压 值	低 压 值
Number	Name	High	Low

要求对高压值和低压值进行判定，组织成血压文件输出，血压输出文件的格式如下：

编号	名字	判定	高压值	低压值
number	name	level	high	low

判定是根据血压值判断出四个等级：

"A"：正常范围
"B"：边界范围
"C"：高血压范围
"D"：立即接受诊断

A，B．C．、D。的划分如下：

A：高压值低于 140 且低于 90
B：高压值低于 140-159 的范围内，而低压值低于 95。或者高压值低于 160，且低压值在 90-94 之间。
C：高压值在 160-179 之间，而低于 120。或者高压低天 180 且低压值在 95-119

之间。
D：高压值大于。等于 180 或者低压值大于。等于 120

[解] 程序中定义两价目记录式文件 fin 和 fout。每次读入一个记录，求出血压的等级，然后写到输出文件中。为了判别血压的等级，程序中定义一个函数 level，它根据高压值和低压值求出等级并返回等级值。

C 语言处理程序如下：

```
#include <stdio.h>
#include <string.h>
#define inrange(x.1,h)((x>=1)&&(x<=h))
FILE*fin,*fout;
Struct in_rec  /*输入文件的记录类型*/
{long Number;  /*编号*/
char Name[20];  /*姓名*/
int High,  /*高压值*/
Low;}  /*低压值*/
struct out_rec  /*输出文件记录*/
{long mumber;  /*编号*/
char name[20];  /*姓名*/
char level;  /*等级*/
int high,  /*最高血压值*/
low;}; /*最低血压值*/
char level(high,low)  /*根据最高血压值和最低血压值求出等级*/
int high,
low;
{if (high<140&&low<90)
return('A')
if (inrange(high,140,159)&&low<95Ⅱ
high<160&&inrange(low,90,94))
return('B')
if (inrange(high,160,179)&&low<120Ⅱ
high<180&&inrange(low,95,119))
return('C')
return('D')
}
main()
{strukct in_rec inrec;
struct out_rec outrec;
fin=fopen("infle","rb");  /*打开输入文件*/
if(fin= =NULL)
{printf("can not create outfile\n");
exit(1);}  /*打开失败后打印出错信息并退出*/
```

```
fout=fopen("outfile","wb"); /*打开输出文件*/
if(fout= = NULL)
{printf("can not create outfile\n"); /*打开失败后,打印出错信息*/
fclose(fin); /*关闭输入文件*/
exit(2);} /*退出*/
while(!feof(fin)) /*当输入文件未结束时*/
{fread(&inrec,sizeof(inrec),1,fin); /*读入一个记录*/
outrec.number=inrec.Number;
strncpy(outrec.name,inrec.Name,20);
outrec.high=inrec.High;
outrec.low=inrec.Low;
outrec.level=level(inrec.High,inrec.Low);别 /*生成输出文件的记录*/
fwrite(&outrec,sizeof(outrec),1,fout);本 /*写一个记录*/
}
fclose(fin);
fclose(fout); /*关闭输入输出文件*/
}
```

2. 直接访问文件

直接访问文件中记录与记录之间没有顺序关系。访问这和文件时，需指出要访问的记录号（即文件中的第几个记录）通常可以通过某种算法根据标识住处直接计算出该记录的记录号，例如可以采用散列法进行计算。所以这灯文件通常又被称为散列文件。

[例 5。5]用 C 语言实现直接访问文件。设某单位有若干职工，其人事档案存放在一个人事档案文件中。该文件中记录的各数据项如下所示：

数据类型	编号	姓名	性别	出生年月			文化程度	职务	工资	住址
				年	月	日				
类型	int	char*10	char	int	int	int	char*10	char*10	float	char*40

该文件按职工的统一编号组织成相对文件，即职工的编号即为该职工在相对文件中的记录号。由于可能发生人事变化，所以该文件中的实记录不一定是连续的。要求编写一人上职工信息查询程序，输入某职工的编号，如果有该职工则打印出职工的姓名。性别。职务和工资信息，否则打印出错信息。

[解]人事档案文件各记录都是定长的。第 I 个记录距文件首的位移为（I-1）*记录长度个字节。因此在访问第三者 I 个记录时，首先利用职权 C 语言中的设置读写位转置函数 fseek,把文件定位到第（I-1）*记录长度个字节的位置，然后利用成块读写函数 fread 即可读入第 I 个记录。

C 语言程序如下：

```
#include <stdio.h>
FILE *fp;
Struct date {int year,month,day;};
Struct person /*职工的类型*/
{int mumber; /*编号*/
char name[10],sex; /*姓名、性别*/
struct date birthday;  /*出生日期*/
char education[10],occupation[10]; /*文化程度、职务*/
float salary;  /*工资*/
char address[40];}; /*住址*/
main()
{struct person p;
int I;
fp=fopen("person","rb"); /*打开职工文件*/
if(fp= =NULL)
printf("file not found\n"); /*打开失败时，打印出错信息*/
{scanf("%d",&j); /*读入职工编号*/
while(I>=1)
{fseek(fp,(long)(I-1)*sizeof(p),0); /*定位到职工记录的开始位置*/
if (fread(&p,sizeof(p),1,fp)>0&&p.number= =i)
printf("%s %c %s %f\n",p.name,p.sex,p.occupation,p.salary);
/*读入职工记录成功并且有该职工时，打印职工的有关信息*/
else printf("no such person\n");
scanf("%d",&j); }
fclose(fp); /*关闭文件*/
}
}
```

3. 索引文件

索引文件（或称随机文件）在结构上除了记录的存储以外，还附有一张索引表，如图 5。1 所示。文件中的记录依次为 R1。R2。。。。。Rn。每次访问文件时不一定从 R 开始，而是根据指定的索引项 L 随机地访问记录 Rk

5.2 文件操作

程序设计语言中通常提供三类基本操作：

⑴文件的建立与删除。

⑵文件的打开与关闭。

⑶文件的读写。有时还提供随机访问操作。

上述基本操作在各相应语言或函数完成。通过这些基本操作，可能组成如下文件操作：

⑴向文件末端添加一个记录。

⑵向有序的议价的中插入一个记录。

⑶在文件中删除一个记录。

⑷修改文件上某个或某些记录。

⑸对文件进行某种统计。

⑹文件检索（将文件上满足条件的记录组成一个新文件）。

⑺合并两个有序文件（合并后的文件仍保持有序）。

⑻文件排序。

⑼生成一个文件夹索引文件。

[例 5.6] 向文件末端添加一个记录。

[解] 设文件的每个记录包含 4 个成分的结构：编号。品名。单价。数量。向文件末端添加一个记录，首先要到达文件末端，这通过调用 fseek 实现，到末端后输入添加的记录.相应程序为：

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#define LEN sizeof(int)
#define LEN sizeof(struct RECORD)
void main (int argi, char ** argc)
{
struct RECORD
{
int id;
char name[20];
float price;
int number;
}r;
int fp;
if (argi< 2)
```

```
{
printf("Uage:ADD_REC filename\n");
exit(0);
}
fp=open(argc[1], O_CREAT | O_WRONLY | O_BINARY,S_IREAD | S_IWRITE);
fseek(fp,0L,SEEK_END); /*文件指针移到文件结尾处*/
printf("Id:");
scanf("%d",&r.id);
printf("name:");
scanf("%s", r.name);
printf("price:");
scanf("%f", &r.price); /*输入 1 个记录在 r*/
printf("number"); /*输入记录写到文件上（尾部）*/
write(fp,&r,LENR);
close(fp);
{
```

设文件名为 rec1.abc,运行该程序（程序执行文件设为 m1.exe）,每次添加一个记录，运行程序的命令行为：m1 rec1.abc.

记录添加完毕后，可运行如下显示文件的程序，其运行程序命令行第 2 个参数为 rec1.abc

[例 5.7 向序文件中插入一个记录。

所谓有序文件指文件的记录按关键字由小到大存放。

为方便起见,设文件的每个记录是一个整数,文件上数据是按值由小到大存放.插入数据设是命令行的第 3 个参数，程序中为 d。若原文件中没有数据，则 d 写入文件；若有数据，则找到第 1 个比 d 大的数据 I，先写入 d，再将 I 和其后各数写回文件中，这也是通过调用 fseek 来实现插入。相应程序为：

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#define LEN sizeof(int)
void main (int argi,char ** argc)
{
int fp, I, d;
if (argi<3)
{
printf("Uage:N1 filename int\n");
exit(0);
```

```
}
d=atio(argc[2]);
fp=open(argc[1],O_CREAT | O_RDWR | O_BINARY,S_IREAD | S_IWRITE);
while(1)
{
if (read (fp,&I,LEN)!=LEN);
{
write(fp,&d,LEN); /*文件已结束，d 添加到文件尾端*/
break;
}
if(I>=d)正   / *文件中读出数据 i 若 i≥d 则先存 d* /
{
do
{
fseek(fp,_1L*LEN,SEEK_CUR); /*文件指针退 1 个记录*/
write(fp,&d,LEN); /*d 写到文件上*/
d=I; /*原 i 作 d，以便修理其它数据*/
} while (read(fp,&I,LEN)= =LEN); /*继续读数据，并判别文件是否完*/
write(fp,&d,LEN);
break;
}
}
close(fp);
}
```

[例 5.8] 在文件中删除一个记录。

[解]仍设文件每个记录是一整数。删除的数据设为 d，删除就是对原文件扫视一遍，值为 d 的记录不写回文件，而不为 d 值的记录均写回文件。相应程序为：

[例 5.9] 修改文件上某些记录

[解] 仍设文件每个记录是一个整数。被修改的数设在 do 中，欲改成的数设有 d1 中。对原文件也扫视一遍。若某个记录的值同 do 的值，则用 d1 的值代替之。相应程序为：

```
# include<stdio.h>
# include<stdlib.h>
# include<.io.h>
# include<fcntl.h>
#defineLENsizeof(int)
void main (int argi ,char **argc)
{
int fp,I ,do ,dl;
if(argi<4).
```

```
{
printf("Uage:N3filename int 0 int 1\n");
exit(0);
}
do=atoi(argc[2]); /*被改数*/
d1=atoi(argc[3]);  / 改正数*/
fp=open(argc[1],o_RDWR|0_BINARY|);
while (read(fp,&I,LEN)= =LEN)
if(I= =d0)
{
fseek(fp,_iL*LEN,SEEK-CUR); /*指针退回 1 个记录*/
write(fp,&di,LEN);  /*改正该数*/
}
close(fp);
}
```

[例 5.10]文件的统计。

设文件的记录包含物资编码、物资品名、单价、数量。这个文件可看成是一个库存台帐，今要求统计物资编码是 5 倍数的那些物资的资金占用。

[解]也是对文件扫视一遍，检查记录中物资编码是 5 倍数时，将单价*数量进行累加，相应程序为：

```
#include<stdio.h>
#inchude<stdib.h>
#include<fcntl.h>
#defoneLENsizeof(int)
#defineLENRsizof(struct RECORD)
void main (int argi,char* * argc)
{
struct RECORD
{
int id;
char name[20]
float price;
int number;
}r;
int fp;
float occupy;
if(argi<2)
{
printf("Uage:N4fiename \n");
exit(0);
}
```

```
fp=open (argc[1],O_RDONLY|O_BINARY);
occupy=0;
while(read(fp,&r,LENR)= =LENR)
if((r.id%5= =0) /*统计条件*/
occupy+=r.price*r,number;
close(fp);
printf("Occupied money is% .2f\n",occupy);
}
```

设文件有如下记录：

```
recl .abc
Id:24 Name :abcdef Price:2.80 Number:5000
Id:15 Name :bcde fghijk Price:20.50 Number:1000
Id:20 Name:xyz price99.90 Number:20
Id:32 Name:ijkmn price5.70 Number:210
Id:25 Name:efgh price7.60 Number:80
Id:45 Name:wvu price6.80 Number:300
Id:50 Name:mu price8.30 Number:800
```

运行程序（n4）的命令行与输出为：

```
n4recl.abc
Occupied money is 31786.00
```

[例 5. 11]文件检索。

[解]将文件上满足某条件的记录检出，并打印输出。设文件如[例 5.10]中每个记录含物资编码、物资品名、单价、数量各项。检出单价数量之积（即占用资金）大于 10000 的物资的品名、单价、数量、占用资金各项，相应程序为：

```
#include<stdio.h>
#inchude<stdib.h>
#include<fcntl.h>
#defoneLENsizeof(int)
#defineLENRsizof(struct RECORD)
void main (int argi,char* * argc)
{
struct RECORD
{
int id;
char name[20]
float price;
int number;
}r;
```

```
int fp;
float occupy;
if(argi<2)
{
printf("Uage:N4fiename \n");
exit(0);
}
fp=open (argc[1],O_RDONLY|O_BINARY);
while(read(fp,&r,LENR)= =LENR) /*检索条件*/
if((r.price*r.number)>10000
printf("Id:%-5dName:%-10s Price:%-10.2f Number:%
-5dPrice*Number:%-10.2f\
nr.id,r.name ,r,price,r.number,r.rpice*r.number");
close(fp);
```

如前例的文件，运行该程序（n5）的命令和输出为：

```
n5 recl.abc
Id:24 Name:abcdef Price:2.80 Number:5000 Price*Number:14000.00
Id:15 Name:bcdefghi jk Price:20.50 Number:1000 Price*Number:20500.00
Id:24 Name:qrstuv Price:88.8 Number:200 Price*Number:17760.00
```

[例 5.12]文件合并。

[解]仍设文件每个记录是一个整数，并且都是以由小到大存放。合并文件需一个中间文件。将两个原来文件看成是两个序列，合并时只需检查两个序列的头，比较较小者先写入中间文件。比较总有一个序列先完，另一序列就将剩余数据抄入中间文件。直到两个文件都结束，合并工作结束，相应程序为：

```
#include<stdio.h>
#inchude<stdib.h>
#include<io.h>
#include<fcntl.h>
#include<sys \stat.h>
#defineLENsizof(int)
void main (int argi,char* * argc)
{
int fp0 ,fp1,fp2,so,s1,io,i1;
if(argi<3)
{
printf("Uage:N6filenamel filename 2\n");
exit(0);
}
fp0=open (argc[1],O_RDONLY|O_BINARY); /*合并的第 1 个文件*/
fp1=open(argc[2],O-BINARY|O_RDONLY); /*合并的第 2 个文件*/
```

```
fp1=open("temp",O_CREAT|O_BINARY|O_WRONLY,S_ IREAD|S_IWRITE);
so=read(fp0,&i0,LEN); /* 文件合并到 temp*/
s1=read(fp1,&i1,LEN);
while((so= = LEN||(s1= = LEN ) ) /*2 个文件中至少有 1 个未完*/
{
if((s1!=NEN)||((S0= =LEN)&&(io<i1))) /*第 2 个文件已结束或第 1 个文件当前记录小*/
{
write(fp2,&i0,LEN); /* 第 1 个文件当前记录写到合并文件上*/
so=read(fpo.&io,LEN); /*再在第 1 个文件上读一个记录*/
continue;
}
if((so!=LEN)||((s1= = LEN)&&(io>il))) /*第 1 个文件已结束或第 2 个文件当前记录小*/
{
write(fp2,&il,LEN); /*第 2 个文件当前记录写到合并文件上*/
s1 =read(fp1,&i1,LEN); /*再在第 2 个文件上读 1 个记录*/
continue;
}
}
close(fp2);
close(fp1);
close(fp0);
remove(argc[1]); /*删除第 1 个文件*/
rename("temp ",argc[1]):/*改合文件为第 1 个文件*/
}
```

[例 5.13]文件的排序。

虽然较少要求对文件排序。但作为一种方法仍是有意义的。

[解]由于文件的记录数不定，并用有时可能很大，因此对文件本身或先倒数组中排序是困难的。今设计一个算法，每次将文件中 num 个记录（num≤100）放到数组 b 中,对数组按关键字排序。将数组 b 与中间文件 templ 合并到中间文件 temp2 中。合并后仍称作 templ。不断从原文件中取一组数到 b ,并合并，直到原文件取完。因此文件排序就变成文件与数组合并。将数组也看成序列，如[例 5.12]一样合并。相应程序为：

```
# include<stdio.h>
#include<stdlib.h>
#include<io.h>
#include<fcntl.h>
#include<sys\stat.h>
#define LEN siz of (int)
#define LENB100
```

```
void main (int argi,char**argc)
{
int fpo.fp1,fp2,so ,s1,I,j;
int b[LENB],num;
if(argi<2)
{
printf("Uage:N7 file name"\n);
exit(0);
}
fpo=open(argc[1].O_RDONLY|O_BINARY);
while((num=read(fpo.b,LENB*LEN)/LEN>0)/*从文件中取num个数到数组b中*/
{
for(I=0;I<num-1;I++) /*对数组排序*/
for(j=I+1;j<num;j++)
if(b[i]>b[j])
{
so=b[j];
b[j]=b[i];
b[i]=so;
}
/ *打开文件 temp1 准备读，打开 temp2 准备写*/
fp1=open("temp1",O_CREAT|O_BINARY|O_RDWR,S_IREAD|S_IWRITE);
fp2=open("temp2",O_CREAT|O_BINARY|O_RDWR|O_TRUNC,S_TREAD|S_I_WRITE)
so=o; /*so 为数组 b 的指针*/
S1=read(fp1,&j,LEN);
While((so<num)||(si= = LEN)) /*数组和文件中都有数未并入 temp2*/
If(s1!=LEN)||((so<num)&&(b[so]<I))) /*文件已完或数组中当前元素较小*/
{
write(fp2,&(b[so]),LEN); /* 数组元素写入 temp2*/
so++;
continue;
}
if((so>=num)||((s1= =LEN)&&(b[so]>=I))) /* 数组已完或文件当前记录较小
*/
{
write(fp2,&I,LEN); /*文件的当前记录写入 temp2*/
s1=read(fp1,&I,LEN);
continue;
}
}
close(fp2);
close(fp1);
remove("temp1");
rename("temp2","temp1"); /*temp2 再改写成 temp1*/
```

```
} /*若原文件未结束，再上去取数，排序、合并*/
close (fp0);
remove(argc[1]);
rename("temp1",argc[1]);
}
```

[例 5.14]生成一个索引文件。

[解] 如 5.13 节文件组织中所述的索引文件（图 5.1），今设原文件的每个记录如[例 5.10]中一样包含物资编码、物资品名、单价、数量各成分。今要生成以单价为关键字的索引文件。其算法是先从原文件中投影出一个以单价和各记录位置的文件。然后对新文件按单价排序（方法见[例 5.13]），排序后的文件即是索引文件，其索引项为单价，相应程序为：

```
#include<stdlib.h>
#include<io.h>
#include<fcntl.h>
#include<sys\stat.h>
#defineLENI sizeof (struct INDEX)
#define LENB 100
#define LENB 100
#define LENR sizeof(struct RECORD)
void main (int argi ,char**argc)
{
struct RECORD
{
int id;
char name[20];
float price;
int number;
}r;
struct INDEX
{
float price;
int offset;
}b [LENB],in
int fpo.fp1,fp2,so,I,j;
int num;
if(argi<3)
{
printf("Uage:N8 recordfilename indexfilename\n");
exit(0);
}
fp1=open(argc[1],O_RDONLY|O_BINARY); /*原文件，下一个是索引文件*/
```

```
fp0=open(argc[2],O_CREAT|O_TRUNC|O_RDWR|O_BINARY,S_IREAD|S_IWRITE);
num=o;
while(read(fp1,&r,LENR)= = LENR) /*投影产生一个单价、位置文件*/
in.price=r.price;
in.offset=num;
num++
write(fp0,&in,LENI);
}
close(fp1);
fseek(fp0,oL,SEEK-SET); /*索引文件的文件指针指到文件开始*/
while ((num=read(fpo,b,LENB*LENI)/LENI)>0 /*文件排序*/
for (I=o;I<num-1;I++)
for(j=I+1;j<num;j++)
if(b)[i].price>b[j].price)
{
in.price=b[j].price;in.offset=b[j].offset;
b[j].price=b[i].price;b[j].offset =b[i].offset;
b[i].price =in.price;b[i].offset=in.offset;
}
fp1=open("temp1".O_CREAT|O_BINARY|RDWR,S_IREAD|S_IWRITE);
fp2=open("temp2",O_CREAT|O_BINARY|O_RDWR|O_TRUNC,S_IREAD|S_IWRITE);
S0=0;
S1=read(fp1,&in,LENI);
While((so<num)||(s1= = LENI))
{
IF((S1!=LENI)||((so<num)&&(b[so].price<in.price)))
{
write(fp2,&(b[so],LENI);
so++;
continue;
}
if((so>=num)||((si= =LENI)&&(b[so].price>in.price)))
{
write(fp2,&in,LENI);
S1=read(fp2,&in,LENI);
Continue;
}
}
close(fp2);
close(fp1);
remove("temp1");
rename("temp2"1","temp1");
} /*文件排序结束/*
}
```

```
close(fp0);
remove(argc[2]);
rename("templ",argc[2]);
fpo=open(argc[2].O_RDONLY|O_BINARY); /*打印输出索引文件*/
while(read(fpo,&in ,LENI)= = LENI)
printf("Price:%-8.2fOffset:%-5d\n",in,price,in.offset);
close(fpo);
}
```

仍以[例 5.10]中所列 recl.abc 文件，生成索引文件 rl.cba.运行该程序的命令行及索引文件如下：

```
n8 recl.abc rl.cba
Price:2.80 offset:0
Price:5.70 offset:4
Price:6.80 offset:6
Price:7.60 offset:5
Price:8.30 offset:7
Price:20.50 offset:1
Price:88.80 offset:2
Price:99.90 offset:3
```

5.3 文件使用用举例

5. 3. 1 C 语言文件使用举例

[例 5.15]仓库入库处理。

[解]散 设仓库帐是一个文件，其每个记录至少包含：物资编码，物资品名、单价、数量各项。入库文件也至少包含上述四项，两记录的结构可以不同，但为方便计，这里设为相同。入库的过程实际上就是文件合并，但与[例 5.12]不同的是对相同物资应同类项合并即两个成分合成一个写到库存文件上去(当然数量应相加）。相应程序为：

```
#include<stdlib.h>
#include<io.h>
#include<fcntl.h>
#include<sys\stat.h>
#defineLENRsizeof(struct RECORD)
void main(int argi.char **argc)
{
structRECORD
{
int id;
char name[20];
float price;
```

```
int numeber;
}ro..rl;
int fp0,fp1.s;
if(argi<3)
{
printf("Uage:N9filename filename \n");
exit(0);
}
fpo=open(argc[1].O_ CREAT|O_B INARY ,S_IREAD |S_IWRITE);
fp1=open(argc[2].O_RDONLY|O_BINARY); /*打开入库文件*/
while(read(fp1.&r1,LENR = =LENR))/*入库文件中取一个记录*/
{
fseek(fpo.Ol,SEEK_SET;)
while((s=read(fpo.*ro.lenr)= = LENR) /*搜索台帐文件*/
if(r0,id= = rl.id ) /*找到相应在的记录*/
{
ro.number+=rl.number; /*修改库存量*/
fseek(fpo.-1L*LENR, SEEK-CUR);
writte(fpo,&ro, LENR); /*存回台帐文件*/
break;
}
if(s!=LENR) /*台帐文件中无此物资（即）新品，则 rl 写入台帐文件*/
write(fpo.&rl,LENR);
}
close (fpo):
close(fpl);
}
```

[例 5.16]仓库出库处理。

[解]设库存文件和相应于单价的索引文件已由[例 5.14]那样产生，今有出库（每个记录的结构包含单价与数量），做出库处理，对出库文件每个记录查索引文件，并根据索引文件找到库存文件的相应记录，然后减去出库数量再存回原库存文件。相应程序为：

```
#include<stdlib.h>
#include<io.h>
#include<fcntl.h>
#include<sys\stat.h>
#defineLENR sizeof(struct RECORD)
#defineLENO sizeof(struct OUTBASE)
#define LENI sizeof(struct INDEX)
void main (int argi,char * * argc)
{
```

```
structRECORD
{
int id;
char name[20];
float price;
int number;
}r;
structOUTBASE
{
float price;
int number;
}o;
struct INDEX
{
float price ;
int offset;
}I;
int fpo,fp1 ,fp2,s;
if((argi<4)
{
printf("Uage:N10 basefilename indexfilename out filename \n");
exit(0);
}
fpo=open(argc[1].O-RDWR|O-BINARY);
fp1=open(argc[2].O_RDONLY|O_BINARY);
fp2=open(argc[3].O_RDONLY|O_BINARY);
while(read(fp2,&o,LENO)= = LENO)
{
fseek(fp1,Ol.SEEK_SET);
while((s=read(fp1,&I,LENI))= =LENI
if(i.price= = o.price)
break;
if(s!=LENI)
printf("no such item in base!\n");
else
{
faeek(fpo.LENR*I,offset *1L ,SEEK_SET);
read(fp0,&r,LENR);
r.number= = o.number;
fseek(fpo,LENR*i.offset*1L,SEEK_SET);
write(fpo,&r,LENR);
}
}
close(fp0);
```

```
close(fp1);
close(fp2);
}
close(fpo);
close(fp1);
close(fp2);
}
```

[例 5.17] 用 C 语言打印文件中含有特定子串的行。从命令行中输入一个文件名和一个子串。找出文件中所有含有该子串的行并打印。

[解] 使用 fopen 打开输入文件。每次从文件中读入一行，使用函数 index 判别该行中是否含有指定的子串。匹配成功的打印出该行。

C 语言程序如下：

```
#include<stdio.h>
FILE*fp;
Getline(str)/*从文件 fp 中读入一行存放在数组 str 中*/
Char *(str)/*从文件 fp 中读入一行存放在数组 str 中*/
Char*str;
{int I;
char c:
I=o;
C=getc(fp);
While (c!='\n' &&c!=EOF)
{str[i]=c;
I=I+1;
C=getc(fp);} /*读入一行*/
Str[i]=0
If(c= =EOF )return(-1);
Else return(i);/*文件结束，则返回-1，否则返回数组 str 中的字符串长度*/
}
index(str,substr)/* 在 str 中查找是否含有子串 substr,有子串时返回子串的位置，否则返回-1*/char str,substr;
{int I,j,k;
for(I=o;str[i]!=o;I++)
{for(j=I,k=o;substr[k]!=o&&str[i]= = substr[k];j++,k++);
if(substr[k]= =0)return(i);
}
return(-1);
}
main (argc,argv)
int argc;
char*argv[ ];
```

```
{char s[1000];
int len;
if(argc!=3)
printf("Usage index filename substrint \n"); /*命令行错，打印使用方法
*/
else
{fp=fopen(argv[1],"r"); /*打开文件失败时，打印出错信息*/
else
{do{
len=getline(s) ; /*从文件中读入一行 s*/
if(index(s,argv[2])>=0)
printf("%s\n",s); /* 如果 S 中含有子串 argv[2]时，打印 s*/
}while(len>=o);
fclose(fp); /*关闭文件*/
}
}
}
```

5．3．2 FORTRAN 语言文件使用举例

[例 5.18]用 FORTRAN 语言统计学生成绩。在文件 STUDENT 中存放了若干学生的学号和语文、数学、外语成绩，格式如下：

数据项名	学 号	语 文	数 学	外 语
类 型	Character*8	I4	I4	I4

要求求出每个学生的学号和总成绩，存放到文件 SCORE 中。

[解]打开文件 STUDENT ，设部件号为 8，打开 SCORE 文件设部件号为 9。每次从 STUDENT 文件中读入一个记录，计算出三门课程的总分并写到 SCORE 文件中。FOR-TRAN 语言程序如下：

```
      CHARACTER NUMBER*8
      INTEGER CHINESE,  MATH，EN G，TOTAL。
*     OPEN(8,FILE='STUDENT',ACCESS='SEQUENTIAL'
      FORM='FORMATTED' ，STATUS ='OLD'),
      OPEN（9，FILE='SCORE'，ACCESS='SEQUENTIAL'，
*     FORM='FORMATTED'，STATUS='NEW'）
10    READ（8，500，END=20）NUMBER，CHINESE，MATH，
      ENG
      TOTAL=CHINESE+MATH+ENG
500   FORMAT(A8,314)
      WRITE(9,510)NUMBER,TOTAL
510   FORMAT（A8，I4）
      GO TO 10
```

```
20   GLOSE（8）
     CLOSE（9）
     STOP
     END
```

[例 5.19]用 FORTRAN 语言随机读文件。在文件 DATAFILE 中，每个记录存放了 3 个 F10.4 格式的实数，要求分别打印出该文件的第 2、4、8、16、32、64 个记录中的三个实数值。

[解]首先以直接方式打开文件 DATAFILE。变量 I 指出要访问记录号，它的初值为 2。读出第 I 个记录并打印,每次 I 乘以 2,如此重复直到 I 大于 64 为止。FORTRAN 语言程序如下：

```
OPEN（8，FILE=DATAFILE，ACCESS=DIRECT，RECT=30，
FORM=FORMATTED，STATUS=OLD
I=2
10 READ（8，500，REC=DA，B，C）
500FORMAT（IX，3F15. 4）
I=I*2
IF（I，LE，64）GOTO10
CLOSE（8）
STOP
END
```

上例中 READ 语句中的说明符 REC=I 指出要读入记录的记录号为 I，执行 READ 语句将该记录读入。

[例 5.20]学生档案查寻。设有两个文件：主文件 MAIN 和索引文件 INDEX。主文件的格式与[例 5.18]中的学生登记表格式相同。索引文件的每个记录有两个域：学号和相应的学生在主文件中的记录号，它的格式如下：

数据项名	学 号	记 录 号
长 度	8B	4B
类 型	I8	I4

索引文件中的记录是按学生学号的升序排列的。主文件和索引文件都是由带格式的记录组成的。

[解]首先把索引表文件读入。以后每次从键盘读入学生的学号，检查学号是否在索引文件中；如果在，则利用 FORTRAN 语言提供的随机文件访问指定的记录并显示该学生的登记表项。

FORTRAN 语言程序如下：

```
SUBROUTINE GETINDEX（NUM，RECNUM，LEN）
INTEGER NUM（1000），RECNUM（1000），LEN
OPEN（8，FILE=INDEX，ACCESS=SEQUENTIAL，STATUS=OLD，）
*FORM=FORMATTED）
I=I
10 READ（8，500，END=20）NUM（I），RECNUM（I）
500 FORMAT（18，16）
I=I+1
GOTO 10
20 LEN=I-1
CLOSE（8）
RETURN
END
SSUBROUTINE SEARCH（N，NUM，LEN，LOC）
INTEGERN，NUM（10000），LEN，LOC
I=1
J=LEN
40 IF（I，GT.1）THEN
LOC=K
ELSE
LOC=-1
ENDLF
RETURN
END
PROGRAM INQUIRE
INTEGER NUM（1000），RECNUM（1000），LEN，SN，AGE，LOC
CHARACTER NAME*20。SEX，ADDRESS*100，DEPARTMENT*40SPECIALITY*40
CALL GETINDEX（NUM，RECNUM，LEN）
OPEN(10,EILE=MAIN,ACCESS=DIRECT,RECL=211,STATUS=OLD),FORM=FORMATTED
50 READ（*，100）N
100 FORMAT（14）
IF（N，LT.0）GOTO60
CALL SEARCH(NUM,LEN,LOC)]
IF(LOC,LT.0)GOTO50
I=RECNUM(LOC)
READ(10,110,REC=I) SN,NAME,SEX,AGE,ADDRESS,DEPARTMENT;SPECIALITY
110 FORMAT(I8,A20,A1,A100,A40,A40)
WRITE(*,120)SN,NAME ,SEX,AGE,ADDRESS,DEPARTMENT,
*SPECIALITY
120 FORMAT(1X ,I8,2H,A1,I2,A100,A40,A40)
GOTO50
60 CLOSE(10)
61 STOP
END
```

习题

5．1 文件拷贝。用 C 语言写一个文件拷贝程序，把源文件拷贝到目标文件，并提示是否需要删除源文件。如需要则将源文件删除。源文件和目标文件名由命令行给出，源文件名在前，目标文件名在后。

程序中说明两个文件：源文件和目标文件。以只读方式打开源文件并建立目标文件。C 语言可能 fopen（文件名，"wb"）,建立一个文件。文件拷贝完成后提示是否需要删除源文件。

C 语言程序如下：

```
# include<stdio.h>
FILE*source,*target;
Main(argc,argv)
Int argc;
Char* argv[ ];
{char c;
if(argc!=3)
printf("Usage dup sourcefile targetfile＼n"); /*命令行不正确时，打印出错信息*/
else
{ ① }
if(source= = NULL)
printf("file%s,not found ＼n",argv[1]; ) /*打开失败时，打印出错信息*/
else
{ ② }
if(target= = NULL)
{printf("can not create file %s＼n",argv[2]);
fcloce(source);}/*打开目标文件失败时，打印出错信息并关闭源文件*/
else
{ ③ }
{c=getc(source);
putc(c,target);
} /*把源文件拷贝到目标文件*/
fclose (source);
fcloce(target); /*关闭源文件和目标文件*/
printf("delete%s?(Y/N)",argv[1]; /*询问是否要删除源文件*/
scanf("%c",&c);
if(c= = 'y'||c= ='y')
{ ④ }
}
}
```

```
}
}
```

5.2 人事档案统计。某单位人事文件格式如下：

数据项名	部门代码	职工代码	年龄
类型	int	Char*6	int

该单位有 10 个部门，部门的代码是 1 到 10。要求输入人事文件，接部门类别和年龄类别统计人数，生成如下统计表。

部门\年龄	<20	20-29	30-39	40-49	50-59	≥60	合计
01	×××	×××	×××	×××	×××	×××	×××
02	×××	×××	×××	×××	×××	×××	×××
…							
10	×××	×××	×××	×××	×××	×××	×××
合计	×××	×××	×××	×××	×××	×××	×××

为了率统计各部门各年龄组的人数，程序中设立二维数组成 table，table[i][j]记录了部门 I 中第 j 个年龄组（0≤j＜6＝的职工人数。每次从人事文件中读入一个职工的记录，根据其部门和年龄修改数组 table。最后打印出数组 table. C 语言程序如下：

```
# include<stdio.h>
FILE*fp;
Struct person /*职工类型*/
{int unif;/*部门*/
char code[6];/*职工代码*/
int age;};/*职工年龄*/
#define MaxUnit 10
int table[MaxUnit][6];
level(age) /*根据年龄确定年龄分组*/
int age;
{if(age<20)return(0)
if(age<30)return(1)
if(age<40)return(2)
if(age<50)return(3)
if(age<60)return(4)
return(5);
}
main( )
{inti,j,k,total;
struct person p;
```

```
fp=fopen("document","rb") ;/*打开职工文件*/
if { ① }
printf("document file not fonnd\n"); /*打开出错时，打印出错信息*/
else
{for(I=o;I<MaxUnit;I++)
for(j=0;I<6;j++)
table[i][j]=o; /*table 初始化*/
total=o;
while { ② }
{if { ③ }
{total=total+1;
I=level(p.age);
Table[p.unit-1][i]++;
}
printf("<20 20-29 30-39 40-49 50-59 >60 total\n");
for(I=o;I<MaxUnit;I=++)
{printf("%8d",i);
k=o;
for(j=o;j<6;j=j++)
{printf("%8d",table[i][j]);
k+=table[i][j];
}
printf("%8d", k);
}
printf("%8d\n",total); /*打印出年龄统计表*/

{ ④ }

}
```

5．3 阅读下列程序说明和程序，填充程序中的{ }

[程序说明]商品文件中记录了商品的有信息，它的格式如下：

数据项名	商品代码	商品名	成 本	单 价	数量	生产厂家
类型	int	Char*20	Double	double	Int	Char*20

要求根据商品更改文件中的内容对商品文件进行更新处理。

（1）如果商品文件中存在和更改文件中相同商品代码的记录，则用商品更改文件中的记录修改商品文件中的记录。

（2）如果商品文件中不存在相同的商品代码，则把更改文件中的商品记录插入商品文件中。

商品更改文件和商品文件中的记录是按商品代码的降序排列的，不存在两个以上相同商品低码的记录。商品更改文件的记录格式与商品文件的记录格式相同。商品文件修改后放在一个新文件中。

[程序]

```
#include<stdio.h>
struct rec
char name[20];
double cost;
price;
int number;
char factory[20];
};
FILE*produtfile,
*modifytile,
*newfile;
main()
{struct rec p,m;
imt i,j;
if ((productfile=fopen("product.old","rb"))= =NULL)
{printf("product file not found\n");
exit(1);}
if((modifyfile=fopen("modify","rb"))= = NULL)
{printf(product file not found \n);
fclose(productfile);
exit(1);}
newfile=fopen("product.new","wb");
I=fread(&p,sizeof(p),1,productfile);
J=fread(&m,sizeof(m),1,modifyfile);
While( ① )
If(p,code>m.code)
{fwrite(&p,sizeof(p).1,newfile);
{ ② }
}
else if({ ③ } )
{fwrite(&m,sizeof(m),1,newfile;
I=fread(&p,sizeof(p),1,productfile);
J=fread(&m,sizeof(m),1,newfile);
J=fread(&m,sizeof(m),1,modifyfile);}
While(I>0)
{fwrite(&p,sizeof(p),1,newfile);
j=fread(&p,sizeof(p),1,productfile);
}
```

```
while(j>0)
{fwrite(&m,sizeof(m),1,newfile);
j=fread(&m,sizeof(m),1, { ③ } );
}
fclose(productfile);
fcose(modefyle);
fclose(newfile);
}
```

5.4 阅读下列程序说明和程序，填充程序中的{ }

[程序说明]有一个学生考试成绩文件，文件的记录格式如下：

数据项名	姓名	成绩 1	成绩 2	…	成绩 7
类 型	Char*20	int	int	…	Int

要按学生姓名的增序打印出学生的姓名和总成绩。

[程序]

```
#include<stdio.h>
#include<strint.h>
FILE*fp;
Struct student
{char name[20];
int score[7];
};
typedef struct node *tree;
struct node
{char name[20];
int total;
tree left,right ;};
tree insert (root,name,total)
tree root;
char root;
char* name;
int total;
{tree t;
if(root= = NULL)
{t=(tree)malloc(sizeof(*t));
strcpy(t->name,name);
t->total=total;
①
return(t);}
elseif(strcmp(root->name,name)>0);
```

```
root->left=insert(root->left,name,total);
else root->right=insert(root->right,name,total);
②
}
print_tree(root)
tree root;
{if (root !=NULL)
{ ③
print f("total score= %d\n", root->total);
④
}
}
main( )
{struct student s;
int I,total;
tree root;
fp=fopen("student","rb");
if(fp= = NULL)
printf("student file not found\n");
else
{⑤
while( ⑥>0 )
{total=o;
for(I=0;r<7;r++)
total+=s.score[i];
root=insert(root.s,name ,total);
}
⑦
fclose(fp);
}
}
```

5.5 文件的随机访问。设有两个文件：主文件 main.dat 和索引表文件 index.dat. 主文件由若干记录组成，每个记录登记了学生的基本情况，包括以下几个数据项：

数据项名	学号	姓名	性别	年龄	家庭住址	系别	专业
长度	8B	20B	1B	2B	100B	40B	40B
类型	char	char	char	int	char	char	char

索引文件中的记录是按学生学号和相应的学生在主文件中的位置。它的格式如下表。

索引文件中的记录是按学生学号的升序排列的。要求读入学生的学号，从索引表文件中找出该学生在主文件中的位置并打印出该学生的所有信息。

数据项名	学 号	位 置
长 度	8B	4B
类 型	char	Long

首先把索引表文件读入。然后读入学生的学号。采用折半检索法检索是否有该学生的记录。如果有，则使用文件定位函数 fseek 把文件定位到指定位置。读出学生的登记表项并打印。C 语言

程序如下：

```
#omclude<stdio.h>
# include<string.h>
#include<malloc.h>
FILE*mainfile,*indexfile;
Typedef struct({char number[8],name[20],sex;
Int age;
Char address[100],department[40],
Speciality[40];}student;
Typedef struct{char number[8]};
Long offset;}index;
Index*ind;
Int len;
Read_index_file()
{index x;
int I,j;
indexfile=fopen("index.dat","rb");/*打开失败时，退出*/
j= ①
len=j/sizeof(index);
ind=("index*"malloc(j));/*为索引表文件分配空间*/
fread(ind,sixeof(index),len,indexfile);/*读入索引表文件*/
fclose(indexfile); /*关闭索引表文件*/
}
search(num)/*在索引表文件中检索编号为 num 的学生，返回学生在索引表文件中的记录号*/
char num[8];
{int I,1,r,comp;
I=1;r=len-1;
While(1<=r) /*折半检索*/
{I=(1+r)/2;
comp= ② ;
if(comp>=0)r=I-1;
```

```
if(comp<=0)1=I+1;}
if(1-r>1) ③
else return(-1);
}
main( )
{char num[8];
student s;
int I;
read_index_file( );/*读入索引表文件*/
mainfile =fopen("main.dat","rb"; ) /*打开文件*/
while(1)
{printf("\ninput student number>");
gets(num); /*读入学生编号*/
if(num[0]= = 'q')break;
else{ ④ ;
if(I<o)printf("student%s not exise \n",num);
else{ ⑤ ;
fread(&s,sizeof (student),1,mainfile);
printf("%s %s %c %d %s %s
%s".s.number.s.name,s.sex.s,age .s,address.s.department,sspeciality);
/*读入学生记录并显示*/
}
}
fclose(mainfile); /*关闭主文件*/
}
```

以下是 FORTRAN 语言程序习题。

5.6 阅读下列程序说明和程序，填充程序中的{ }

[程序说明]在磁盘上有一个存放学生成绩的文件，文件名为 SCORE，文件的记录格式如下：

数据项名	学生姓名	学号	语文	数学	物理	化学	外语
类型	Char*10	I4	I3	I3	I3	I3	I3

要求建立一个名为 TOTAL 的文件，它的记录格式如下：

数据项名	学生姓名	学号	语文	数学	物理	化学	外语	总成绩
类型	Char*10	I4	I3	I3	I3	I3	I3	I3

按学生总成绩的降序排列，并依次打印出学生的学号和总成绩。

[程序]

```
SUBROUTINE SORT （TOTAL，INDEX，N）
INTEGER TOTAL（2000），INDEX（2000），N
DO100 I=1，N
100 INDEX（I）= ①
DO110 I=1，N-1
DO110J=I+1，N
IF（ ② ）THEN
K=INDEX（I）
INDEX（J）=K
ENDIF
110 CONITNUE
RETURN
END
CHARACTER*10NAME（1000）
INTEGERCHINESE（1000），MATH（1000），PHY（1000），CHEM（1000）。
*ENG（1000），TOTAL（1000），NUM（1000），I，INDEX（1000）
OPEN（8，FILE =FORMATTED，STATUS =OLD）
OPEN（9，FILE=TOTAL，ACCESS =SEQUENTIAL），
FORM=FORMATTED，STATUS=NEW）
I=1
READ（8，500，END=20）NAME（I），NUM（I），CHINESE（I），MATH（I），
PHY（I），CHEM（I），ENG（I），TOTAL（I）
500 FORMAT（A10 ，I4，5I3）
④
WRITE（*，*）NAME（K），NUM（K），TOTAL （K）
25 CONTINUE
510 FORMAT（A10，I8，6I5）
511 CLOSE（9）
512 STOP
513 END
```

5．7 阅读下列程序说明和程序，填充程序中的

[程序说明]某公司职工收入文件中存放了该公司职工从 1 月份到 12 月份的工资收入，格式如下：

数据项名	职工编号	姓名	工作部门代码	1 月份工资	…	12 月份工资	奖金总额
类型	16	char*10	13	F6.2	…	F6.2	F6.2

要求计算出每个职工应缴的所得税和所有职工所缴的所得税总和.所得税税率计算方法如下

税率=

1% 年收入≤3000
1.5% 年收入为3000~5000
2% 年收入≥5000

[程序]

```
CCHARCTER*10NAME
INTEGER NUM, CODE
OPEN (8, FILE ='SELARY', ACCESS='SEQUENTIAL'
FORM='FORMATTED', STATUS='OLD')
{ ① }
READ(8,500, { ② }) NUM, NAME, NODE, S1,S2,S3,S4,S5,S6,S7,S8,
S9,S10,S11,S12,BOUND
T= S1+S2+S3+S4+S5+S6+S7+S8+S9+S10+S11+S12+BOUND
IF ( T.LR.3000.0 ) THEN
TAX=T*0.01
ELSE IF (T.LT.5000.0) THEN
TAX=T*0.15
ELSE
TAX=T*0.02
ENDIF
WRITE (*,510) MANE,TAX
{ ③ }
GOTO 10
20 CLOSE(8)
{ ④ }
500 FORMAT(I6,A10,I3,12F6.2)
510 FORMAT(1X,A10,4HTAY= ,F10.2)
520 FORMAT(1X,10HTOTAL, TAX=,F10.2)
STOP
END
```

第 6 章 事务处理流程图

6.1 概 述

6·1·1 事务与事务处理

1．事务处理与事务处理系统

事务：事务是具有特定目标的任务，它通常联系企事业单位中的管理工作。事务可大同小，但必须具有"将定目标"。例如，库房管理中的"入库"是一个事务，其目标就是记录检验过的货物已进入仓库成为库存。这样的特定目标应该是明确的，表达应该是简洁的。

事务处理；事务处理是完成事务的动作。因此事务处理应服务于该事务的"特定目标"。它说明怎样完成"特定目标"所规定的一系列要求。例如，"入库"事务处理应完成：①登记进入仓库的货物（名称、规格、单价、数量、产地等）及位置（仓位）；②由于库存增加而修改库房占用流动资金的数额；③计算库存是否超限等。

事务处理系统：事务处理系统为一组事务处理的有机组合，它具有下述特点：

（１）系统性和特定的系统目标。

（2）所含一组事务，正好能覆盖系统目标。

（3）每个事务既有一定独立性，相互间又有一定联系，这种联系是通过数据进行的。

例如，将库房管理作为一个事务处理系统。它包括入库、出库、库存查询与分析三个事务。（１）其系统地反映在三个事务按一定关系形成一个整体，并具有特定的目标：对货物出、入库进行管理，并对库存进行有效分析。（2）所列三个事务正好覆盖系统目标。（３）库、出库、库存查询与分析都具有一定独立性，相互间又有一定联系。

2．事务处理对象

事务处理的对象是信息，信息是赋予约定意义的数据。数据位于现代事务处理的中心现代化的管理以数据为依据。所有事务处理都可以看作是在一组数据集上的操作。

这里所述数据不仅是数，还包括字符、图形、语言文字，诸如姓名、颜色、真假一类的概也都可作为数据被处理，甚至报表、文件、台帐、各种凭证、电报、传真等也可作为数据被处理。
数据是事等处理的依据，也是事务处理的结果。例如，入库事务，处理对象有入库单（凭证）、日或月入库文件、库存文件（台帐）。

3．事务处理流程与数据流程

事务处理流程与数据流程有紧密的联系，事务处理流程是事务（或事务分解的一组动作）之间相互关系及处理的先后次序的表示。如前所述，数据是事务的处理依据，也是事务的处理结果。一个电务使用一些数据，经处理产生另一些数据. 所产生的数据与另外一些数据又为另一个事务所使用，并产生另一些数据。这样，一些数据经处理成新的数据，它们再经处理又形成另一些数据，这就形成数据流程。

处理流程侧重于处理功能的描述，数据流程侧重于数据流动与演变的描述。数据流程中的会合点或演变点即是处理节点。因此，事务处理流程与数据流程应该是一致的。

例如，图 6．1 是库房管理的流程图，其中矩形框是处理节点，其它均为数据。

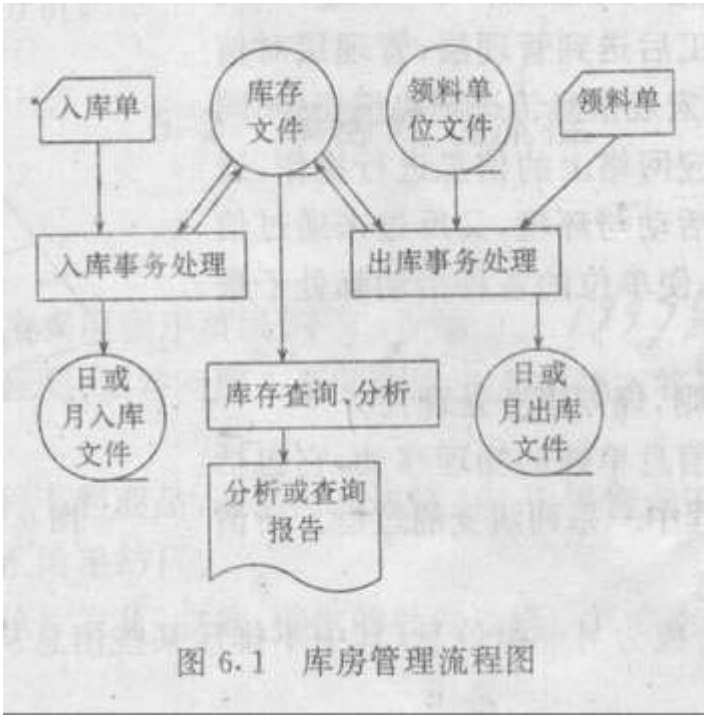


图 6.1 库房管理流程图

6．1．2 事务处理模型

事务处理或事务处理系统在客观世界中大量存在。但是计算机化的事务处理系统并不是实际系统的翻版。在设计一个计算机事务处理系统时，应对实际系统进行抽象，即通过如下步骤定义事务处理模型。

1．抽象与定义信息

信息是人们对客观世界认识的结果。人们自觉或不自觉地认不间断地接受信息、消化处理信息，并发送新的信息。而应用计算机技术接受、处理、发送信息时，首先应从客观世界抽象信息，并予以定义。

一般生产单位处理的信息，主要涉及人、财、物三个方面。所有信息都是这三方面客观存在的映射，它们反映了人、财、物三方面的变化，相互差异与关系。因此，这些单位的信息抽象与定义，应从人、财、物的客观存在与活动出发。

2．描述信息流，定义信息网络

通常都说企业中存在四条流：人流、物流、资金流和信息流。实际上，信息流是另外三条流客观存在与活动的映射。例如，物流反映了从原材料入库到产品销售的流程，有入库、出库、加工、在制品、装配、产品、销售等环节。与物流相对应的信息流以凭证、台账、报表等形式进行流动。反映物流的状况，有入库单、领料单、库存台账、加工进度、加工工时消耗、加工质量、加工设备与能源情况、在制品库存情况、产品库存情况等信息。管理人员是通过信息流来了解与控制物流的。因此，信息流的完备与流畅程度反映了一个企业的管理水平，也是设计事务处理系统的重要基础工作。

信息是应充分利用的一种重要资源，信息流应能使各层次管理人员满足关于信息的需求，因而信息流的描述是事务处理设计中十分重要的一部分。

企事业单位的管理通常分三个层次：决策层）管理层，操作层。各条信息流及各信息流的纵横关系形成一个信息网络。不同层次管理人员关于信息的需求不同，决策层高层管理人员需要宏观的信息，以便进行辅助决策或规划；管理层的中层管理人员应对各专业信息进行详细操作；操作层的有关人员则是采集信息并进行录入与初加工，因此这样的信息网络呈立体形（如图 6．2）。信息从操作层采集初加工后送到管理层；管理层对信息进行专业加工，汇总成宏观信息传送到决策层。不同层次的管理人员利用相应网络上的信息进行操作，通过信息网络了解单位的活动与环境，又反过来通过信息网络控制单位的活动，使单位的各种活动都处于最佳状态。



图 6.2 信息网络金字塔

3．研究信息演变规则，规划事务处理任务.

所谓信息流并不是信息单纯的物理移动,它包括了信息从产生到归宿过程中一系列演变的过程。所谓"演变"包含两重含义：

（1）一组信息经会合成为另一组信息（其中不排斥某些信息从前一组延伸到下一组的情形）。

（2）一组信息引发产生另一组新的信息。

信息的演变与信息属性有关。从信息资源生命周期看，事务处理的信息有四类：

计划类，描述有关计划的信息。

业务类，执行计划时产生的各种信息。

状态类，描述执行计划过程中特定断点状态的信息。

分析类，描述执行计划的结果及有关趋势的信息。

信息的演变就是这四类信息周期性地进行变化。即首先有计划类信息；由计划引出业务类信息；又由于业务，随之产生状态类信息；对状态进行分析，得到分析类信息;分析类信息反过来又影响并产生新的计划类信息,使之进入下一个周期。整个演变过程可称之 PTSA 循环。其中 P(Plan)表示计划类信息,T(Transaction)表示业务类信息，S（State）表示状态类信息，A（Analysis）表示分析类信息。

PTSA 循环每完成一个周期，便产生新的信息，并进入下一个周期，形成了信息的流动。客观上反映了管理活动的推进，这种活动可由图 6．3 表示。

引进 PTSA 循环目的是揭示事务处理中信息演变的规则，以便掌握信息处理的规律。

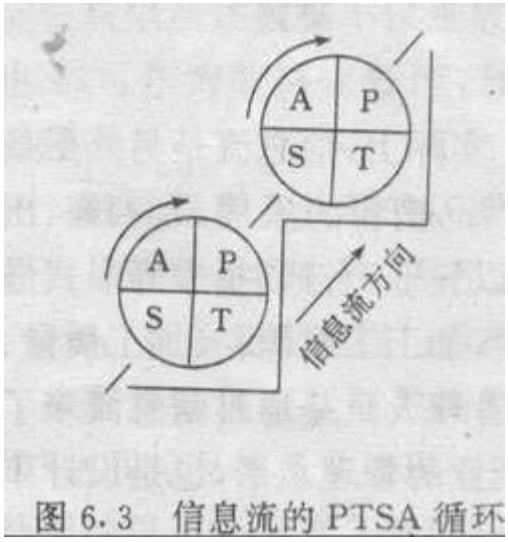


图 6.3 信息流的 PTSA 循环

4．信息的存储原则

信息的存储应考虑到避免冗余及适应事务处理的变化这两方面。避免冗余，并不是键厅一切冗余，有时为了提高处理效率允许部分冗余 适应事务处理的变化，就是充分利用数据库技术，进行信息储存设计。即数据库结构的设计不是手工处理的翻版，而应建立在数据库理论之上。

5．定义事务处理模型

根据上述抽象与定义的信息网络和数据库布局与结构的考虑，说明有哪些加工、处理、变换的节点及这些节点的操作规范，从而定义了一个事务处理模型。一个模型应包括：信息网络的结构与存储的规格说明，所包含的功能节点及功能的操作规格说明等。在下面 6．3 节中将结合实例予以具体分析。

6.2 事务处理流程

6.2.1 例析

[例 6.1]1990 年度高级程序员级（下午）试题二。

阅读下列说明和流程图，回答问题 1 至问题 3，把解答填入答卷的对应栏内。

[说明]

某毛纺厂生产 500 种毛料商品，这些商品送到 300 个销售点销售。销售点应在收到商品后的规定时间内把货款汇给毛纺厂。

流程图 6.4 描述了该厂发货、收款、催款的处理过程。其中商品文件和销售点文件的记录格式如下：

商品文件

商品代号	商品名称	单价

销售点文件

销售点代号	销售点名称	地址

发货单的格式如下

发出日期	销售点代号	商品代号	数量	金额

:收款单的格式如下:

收款日期	销售点代号	商品代号	数量	金额	该商品的发出日期

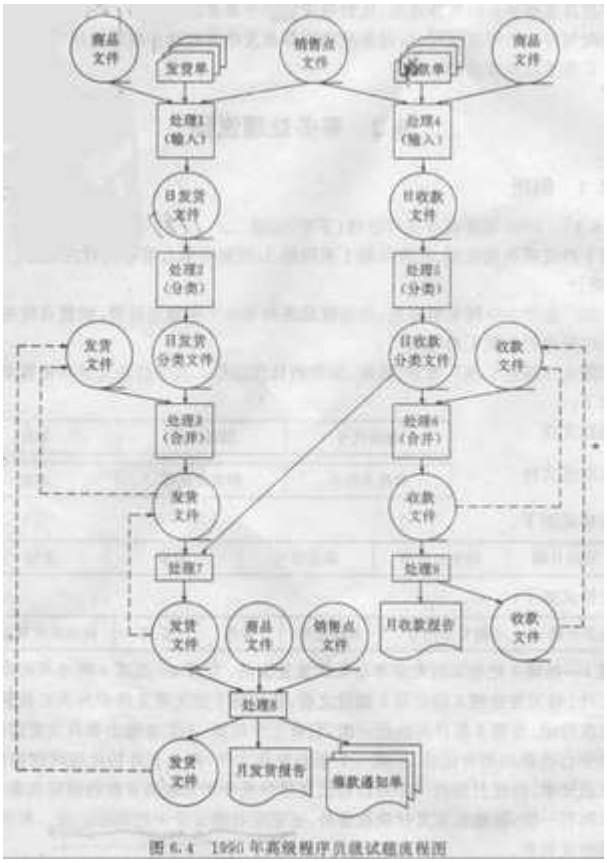


图 6.4 1990 年高级程序员统试题流程图

处理 1-处理 3 把当天的发货单合并到发货文件。处理 4-处理 6 把当一的收款单合并到收款文件。每天在处理 3 和处理 5 做过之后，由处理 7 在发货文件夹中当天已收款的记录上，加上已收款标记，处理 8 在月末执行一次，它有三个功能：①汇总输出本月发货清单;②删除发货文件中已收款的所有记录,形成一个新的发货文件,作为下月初处理时的初始文件;③产生催款通知单,以便对对联些一个月以前已发货但至今仍未收到货款的销售催款。处理 9 也每月末执行一次，除输出本月收款报告外，还删除收款文件中的所有记录。现假定不会有完全相同的发货单。

[问题 1]指出流程图 6.4 中应在哪几个处理框中检查发货单和收款单的错误,并分别指出它们各能指出什么错误。

[问题 2]如果把流程图 6.4 中从日收款分类文件到处理 7 的连线改成从日收款文件到处理 7 的边线,则有什么缺点,理由是什么?

[问题 3]如果把流程图 6.4 中从日收款分类文件到处理 7 的连线改成从收款文件到处理 7 的连线,则有什么缺点,理由是什么?

该题描述了销售管理中的三个"事务"。发货与收款这两个事务是单独处理的，另外还有一平衡处理事务。

在发货事务中，处理的依据是：发货单、销售点文件、商品文件。处理的要求是：货物发出要正确；发出货物要有记录。于是反映在流程图上"处理 1"，应检查上发货单是否正确，这是试题中问题 1 要求的。检查发货单是否正确是与发货单格式有关。检查上面五个项目是否合法以及金额与数量是否一致（单价×数量=金额）。

经处理 1 检查发货单正确后，输出日发货文件，以记录发货情况。

由处理 1 产生的"日发货文件"是按时间顺序的流水账，这对收款结算是不方便的。于是将它按销售点代号分类，这由处理 2 完成，目的是为了提高处理效率。

由于货款不是当日付清，于是应有总的发货文件，以便对应付款结算。故处理 3 合并日发货分类文件到发货文件中去。显然，发货文件是经过分类的，并且该文件应是常存的。而日发货文件与日发货分类文件都是临时的。

同样，收款事务处理的依据是：收款单、商品文件与销售点文件。处理要求是：检查收款单的正确性与汇总日收款分类文件及收款文件，以入每月报收款报告。其中处理 4 应检查收款单的正确性,检查方法是参照收款单格式检查前面五项合法性与正确性。其分类与合并的目的是：分类是为效率，合并是为月报。

平衡发货与收款的处理由处理 7 及处理 8 完成。处理 7 完成发货文件上登录"已收款"标记，作已收款标记的条件是：发货文件与日收款分类文件上销售点代号、商品代号、商品发出日期、数量与金额应该一致。显然，这里应检查：重复付款、找不到相应发货记录、商品数量与金额方面不一致等错误。

由于发货与收款两个事务是分别处理的，最后进行汇总平衡，两者会合点如倾听设置，原题在处理 7。即处理 7 的处理依据是：发货文件与日收款分类文件。会合点能不能变化，可以有几种会合方式：

（1） 日发货分类文件与收款文件会合。因为发货在先，收款在后，因此不能这样会合。

（2） 发货文件与收款文件会合。这样做有两方面缺点：一是处理 7 每天执行一次，收款文件由于包含已处理过的收款记录，因而降低了处理 7 的效率，增加不必要的开销。二是若处理 7 每月执行一次，就不可能及时纠正当天收款单的错误。这个会合也不合适。

（3） 发货文件与日收款文件会合。由于日收款文件没有分类，也将影响处理 7 的速度。
因此试题中将发货文件与日收款分类文件会合是合适的。

上述（1）（2）（3）回答了试题中问题 2 与 3。

6.2.2 事务处理流程式要点

1．1987 年-1995 年试题综述

1987 年到 1995 年有关事务处理流程的试题共 17 题。其中：销售管理方面的题 4 例；生产管理方面的题 3 例；电信计费方面的题 2 例；经济信息统计方面题 2 例，文件合并题 2 例（实为 1 题）；库房、工资、考务、图书编目的题各 1 例。

表 6.1 历年试题涉及问题表

试题\关系\问题	a	b	c	d	e	f	g	h	i	j	k	l
1987 年试题一		√										
1987 年试题二						√	√					
1987 年试题三	√			√	√		√			√		
1988 年试题一	√		√	√	√	√	√	√				
1988 年试题二			√		√		√		√			
1989 年试题三	√		√					√				
1989 年试题一				√	√		√					

1990 年试题二	√		√	√								
1990 年试题三	√			√	√						√	
1991 年试题三					√							√
1991 年试题四	√		√				√					
1991 年试题五					√						√	
1992 年试题一					√	√	√					
1993 年试题一			√		√		√					
1994 年试题一			√		√						√	
1994 年试题四			√			√	√					
1995 年试题一	√				√	√						

在这 17 例的题中，提出的问题主要可分为：

（a） 完善处理流程，或指出处理内容，或指出处理结果。

（b） 指出流程图中错误，或为避免错误应在某处添加某处理项目。

（c） 为某目的需改哪些处理，或改变处理会引起什么后果。

（d） 指出错误清单内容。

（e） 文件的记录应包含哪些内容。

（f） 处理需什么样文件，或文件有什么特征，或引入某文件有什么好处。

（g） 文件怎样分类，或指出关键字。

（h） 某处理的频度。

（i） 输入数据的格式。

（j） 题意中的分类有何好处。

（k） 为某目的适当修改文件的记录内容。

（1） 指出图中数据流名，或指出哪些位置数据可增加或删去。

从上表可以看出，提出问题最多的是（e）即关于文件记录的内容。其次是文件的分类排序（g），再次是当处理改变或某目的应改哪些处理的（c），及完善处理（a）。
从趋势来看，，事务处理从简单的分类、合并、统计等处理发展为数据流分析与文件的关系，即逐步向系统发展，亦即逐步由文件系统的内容向数据库系统的内容发展。

2．事务处理流程要点

事务处理流程图一般由若干处理与若干数据组成，在评估流程式图并回答试题总是问题时，应该注意下述一些问题。

从"处理"的角度来说，必须注意：

（1） 每个事务处理均有一个特定目标，这一目标往往都是文字说明的。与此对应的处理应能覆盖所设定的目标。例如，前述 1990 年度高级程序员下午试题二的目标是发货、收款、催款。其中处理 1、4、7、8 覆盖了问题的上目标。因此，对于说明处理要求，都应从问题目标进行考虑。

（2） 除覆盖问题目标的处理外，还有两类处理应予考虑：一是为了处理的正确性，设计一些处理框，以检查输入数据的数据项及数据的值域；另一种是检查数据会合时数据的一致性。例如 1990 年度高级程序员下午试题二中，处理 1、4 分别检查发货单、收款单的正确性；而处理 7 检查发货文件与日收款分类文件中相关数据的一致性。二是为了处理的效率，如速度、次数、减少处设防问等而引进了一些处理。

（3） 在一个流程式图内，在一条流程上的各个处理不能有功能上的重复。如有重复，不是有错，就是流程还可优化。

（4） 每个处理总有处理的依据--使用数据，以及处理结果--产生数据。例如，1990 年度高级程序员下午试题二中，处理 1 是发货处理，其依据发货单是显见的，但还需商品文件，这是为了能查出发货单中商品是否为所储商品，该商品是否满足该单发出；另一方面，还需要考虑发货对象，即是否注册的销售点。因此还需销售点文件。而作为处理 1 产生的数据，是日发货文件。使用或者产生数据与处理相互匹配是十分重要的。

从数据的角度来说，必须注意：

（1） 注意流程图最初的输入数据与最终输出数据，考虑从输入到输出之间数据演变的情况。例如，1990 年度高级程序员下午试题二，最初；输入为发货单、收款单、输出为月发货报告、月收款报告及催款通知单。从输入到输出数据演变，

由于发货与收款单独处理，于是数据演变为：发货单-发货文件-月发货报告；收款单-收款文件-月收款报告；收款单--收款文件--月收款报告；而发货文件与收款文件会合有催款通知单。根据数据的演变与流程，关于从输入到输出应有哪些数据就比较清楚了，其作用也可以从演变方面了解。

（2） 考虑数据流程中，哪些数据应作为文件形式出现，哪些是中间使作的临时数据。在数据演变中，一些数据经多个"处理"加工后得到结果，每加一次就产生一个新数据，对这些数据分析，就能得出各数据的存储要求。

（3） 对数据按问题要求设计数据结构。输入/输出数据的结构与问题有关，而中间数据的结构除与输入/输出数据有关外，还与处理有关。例如，上述举例的试题，仅考虑发货、收款、催款，并没有考虑发货时工厂有没有这些商品库存，也没有考虑销售点的能力，是否可发这些货。因此，在商品文件中并不标记库存，在销售点文件中并不标记最大销售量。发货文件虽然类似发货单的格式，但考虑支该笔发货有没有付款，故结构上还应增加一数据项，以表示付款与否。

（4） 为减少数据冗余，保证数据一致性，数据文件设计中应考虑关系，亦即各种文件的记录之间的联系。

6.3 事务处理流程图设计举例

6.3.1 流程图设计较

每个企业按人、财、物的信息资源，可划分为若干系统任务，每个任务对某种信息资源进行管理。虽然不同任务管理的对象与管理的模式不同，但其流程图设计原理是一样的。本节以物资供应这一任务为例设计相应的流程图。

1. 决策层管理人员对于物资供应信息的需求

决策层管理人员关于物资历供应的目标是：保证生产、压缩资金占用。

决策层管理人员控制物资供应活动，必须掌握物资借供应的四类周期性变化验的信息，他们的需求可见表面 6.2

表 6.2 决策层关于物资供应信息需求

信息类 / 任务	计划	业务	状态	分析
物资供应	物供计划 用款计划	合同统计 物资行情	库存情况 资金占用情况 物资齐套情况	物耗统计.分析 材料利用率统计分析 A 类物资控制情况 资金周转统计分析 材料质量统计分析

以上信息，应成为物资供应流程的输出或所能访问的文件。

2. 物资历供应流程图

物资供应的目标是保证生产、压缩资金占用，这就是说，物资供应应根据企业生产经营的计划，按时提供必要的物资，但又不能扩大库存，以压缩资金占用，按照对信息资源管理的周期：产生一获取一服务一归宿来考虑供应的事务。这些事务应能覆盖物资供应的目标。表 6.3 列出了为这一层分解出的 6 个任务。

表 6.3 物供信息管理的任务分解

周期 / 任务	产 生	获 取	服 务	归 宿
物资供应	计划管理	合同管理	仓库管理 齐套管理	统计分析 计划价格管理

计划管理是将企业生产经营计划转化为物资供应的有关计划；合同管理便落实这些计划，获取所需物资；获得物资后便有保管，另外考察是否满足生产需要，应有仓库管理及齐套管理；最后统计分析以了解物资消耗、利用情况、资金占用情况；计划价格管理是由于原材料价格波动而通常设计的。还可有一些事务，但它们是次要的，或由于计算机化管理而成为不必要的。

对于上述 6 个事务，每个事务的目标分别为：

计划管理：根据企业生产计划与资金情况，编制物资供应计划、物资储备计划以及采购计划、用款计划。

合同管理：根据计划，采购落实合同，对合同有效管理，跟踪物资在途、入库情况，进行合同统计，并对退货、理赔作出报告。

仓库管理：有效管理材料入库、出库以及物资存放，并对库存进行有关分析统计。

齐套管理：按生产计划及工作令，保证生产氙需用资，对材料缺料作出局快速反应。

统计分析：进行常规统计分析工作（如物耗、利用率、资金周转周期、材料质量等）。

计划价格管理：编制内部计划价格。

为达到上述目的，可对每个事务再分解成若干功能，要求每组功能可覆盖相应事务的目标。每个功能将对应流程式图上一节点。表 6.4 为事务分解表。

表 6.4 事务分解表

事 务 名	功 能 名
计划管理	编制物供计划与储备计划 编制采购计划与用款计划 计划执行的考核
合同管理	采购 合同审核与登录 合同执行 合同统计与检查 货单审核与登录
仓库管理	检验入库 库存管理 限额发料 库存统计与分析
齐套管理	齐套管理
统计分析	统计分析
计划价格管理	编制计划价格

每个功能的目标是明显的。为完成功能的目标，必须分析该功能的处理依据及处理结果。处理依据即该功能的输入，就是使用数据类。处理结果即功能的输出，就是产生数据类。每个功能都可以看成将输入转换为输出的变换。所谓流程图就是这一系列变换的动态描述。为便于分析/综合，将每个功能使用与产生的数据类列表，即所谓"功能-数据类"表（表 6.5），由该表可以方便地描述有关的流程式图。

表 6.5 物资供应的"功能-数据类"表

编号	功能名	使用数据类	产生数据类
P10	编制物供计划与储备	生产经营大纲文件 生产计划文件 材料定额文件 领导指示	物供计划文件 储备计划文件
P02	编制采购计划与用款计划	物供计划文件 储备计划文件 库存情况文件 资金情况文件 市场行情文件 临时生产计划文件 缺料情况文件 超储物资文件 资金占用情况文件	采购计划文件 用款计划文件
P03	计划执行的考核	采购计划文件 采购执行文件 物供计划文件 物资完成情况	各类计划考核表 物供计划执行文件 储备计划文件

		文件 缺料情况文件 储备计划文件 储备统计文件	
P04	采购	采购计划文件 市场行情文件	合同
P05	合同审核与登录	采购计划文件 合同	合同文件 采购执行文件 非法合同文件
P06	合同执行	合同文件 （日）货单文件 （日）入库文件 不合格品文件	合同执行文件
P07	合同检查与统计	合同文件 合同执行文件	已完成合同副本文件 在途物资统计文件 理赔、退货文件 合同统计文件
P08	货单审核与登录	合同文件 货单 采购计划文件	（日）货单文件 货单文件 非法货单文件
P9	检验入库	货单文件 检验单	（日）入库文件 入库文件 不合格品文件
P10	库存管理	（日）入库文件 （日）出库文件	库存文件
P11	限额发料	库存文件 物代计划文件 领料单	（日）出库文件 发料单 出库文件
P12	统计与分析	库存文件 出库文件 物供计划文件 储备计划文件 合同统计文件	库存情况文件 储备统计文件 超储物资文件 A 类物资情况文件 资金占用情况文件 价格统计文件 物耗统计分析报告 实耗统计分析报告 材料利用率分析报告 资金周转统计分析
P13	齐套管理	生产计划文件 临时生产计划文件	物资完成情况文件 缺料情况文件 催货文件

		工作令 库存文件 出库文件 在途物资统计文件 合同文件	
P14	市场行情收集	合同统计文件 物资文件 情况汇报（含情报网上信息）	市场行情文件
P15	计划价格管理	市场行情文件 价格统计文件	计划价格文件
P16	用款管理	用款计划文件 采购执行文件 合同执行文件	用款台账文件

其中，每个功能将"使用数据类"作为依据，经过处理后形成"产生数据类"。使用数据类有的其它任务传递而来，而产生数据类，又向其它任务进行传达室递。

3. 物资供应有关的数据类词典（表 6.6）

表 6.6 数据类词典(物供)

编号	数据类名称	相关功能		流向外部
		(产生)	(使用)	
d01	物供计划文件	01	02、03、11、12	决策层
d02	储备计划文件	01	02、03、12	决策层
d03	采购计划文件	02	03、04、05、08	
d04	用款计划文件	02	16	
d05	各类计划考核表	03		
d06	物供计划执行文件	03		
d07	储备计划执行文件	03		
d08	合同	04	05	
d09	合同文件	05	06、07、08、13	
d10	采购执行文件	05	03、16	
d11	非法合同文件	05		
d12	合同执行文件	06	07、16	
d13	用款台账文件	16		

d14	已完成合同副本文件	07		
d15	在途物资统计文件	07	13	
d16	理赔、退货文件	07		
d17	合同统计文件	07	12、14	决策层
d18	（日）货单文件	08	06	
d19	货单文件	08	09	
d20	非法货单文件	08		
d21	（日）入库文件	09	06、10	财务
d22	入库文件	09		
d23	不合格品文件	09	06	质量
d24	库存文件	10	11、12、13	
d25	（日）出库文件	11	10	财务
d26	发料单	11		
d27	出库文件	11	12、13	
d28	库存情况文件	12	02	决策层
d29	储备统计文件	12	03	
d30	超储物资文件	12	02	
d31	A 类物资情况文件	12		决策层
d32	资金占用情况文件	12	02	决策层
d33	价格统计文件	12	15	
d34	物耗统计分析报告	12		决策层
d35	实耗统计分析报告	12		决策层
d36	材料利用率分析报告	12		决策层
d37	资金周转统计分析	12		决策层
d38	物资完成情况文件	13	03	决策层
d39	缺料情况文件	13	02、03	
d40	催货文件	13		
d41	市场行情文件	14		
d42	计划价格文件	15	02、04、15	

使用物资供应以外任务的数据类称为"相关外部数据类"，表 6.7 表示这些数据类。

表 6-7 相关外部数据类词典(物供)

编号	数据类名称	相关功能	来源
e01	生产经营大纲文件	01	计划任务
e02	生产计划文件	01,13	生产任务
e03	材料定额文件	01	设计任务
e04	领导指示	01	辅助决策任务
e05	资金情况文件	02	财务任务
e06	临时生产计划文件	02,13	生产任务
e07	工作令	13	生产任务
e08	货单	08	外单位
e09	检验单	09	质量任务
e10	领料单	11	生产任务
e11	物资文件	14	国家文件
e12	情况汇报	14	各有关人员

4．流程图

物资供应的流程图可从表 6.5 产生。每个功能即为一个得理框，使用数据类与产生数据类就是其输入与输出。按表即可得物资供应流程式图 6.5(a)至图 6.5(e)。

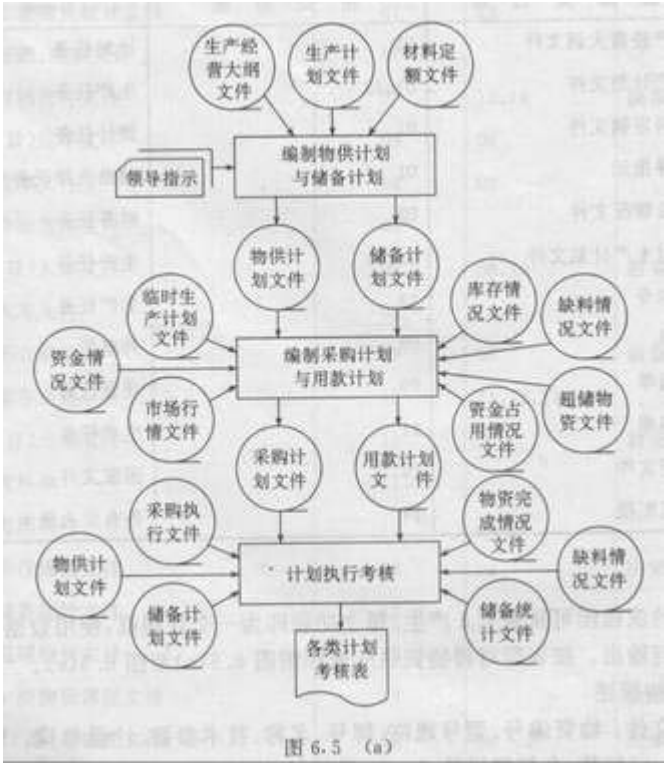


图 6.5 (a)

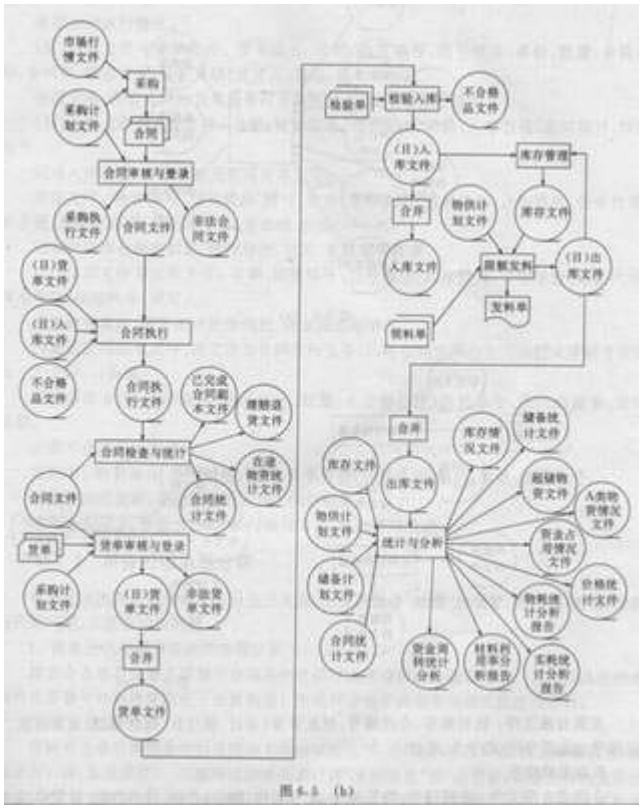

市场行情文件
采购
采购计划文件
合同
合同审核与登录
采购执行文件
合同文件
非法合同文件
(日)货单文件
(日)入库文件
合同执行
不合格品文件
合同执行文件
已完成合同副本文件
理赔退货文件
合同检查与统计
在途物资统计文件
合同文件
合同统计文件
货单
货单审核与登录
采购计划文件
(日)货单文件
非法货单文件
合并
货单文件
检验单
检验入库
不合格品文件
(日)入库文件
库存管理
合并
物供计划文件
库存文件
入库文件
限额发料
(日)出库文件
发料单
领料单
合并
库存文件
出库文件
物供计划文件
储备计划文件
合同统计文件
统计与分析
库存情况文件
储备统计文件
超储物资文件
人员物资情况文件
资金占用情况文件
价格统计文件
物耗统计分析报告
资金周转统计分析
材料利用率分析报告
实耗统计分析报告

图 6.5 (b)

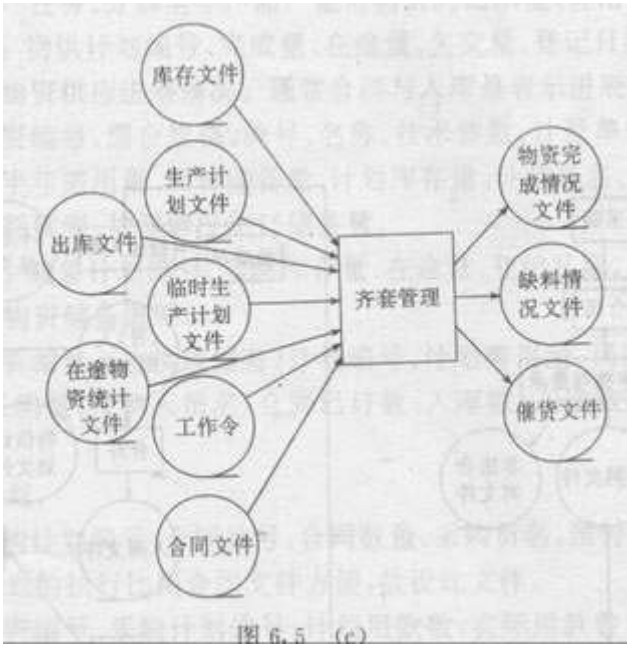

库存文件
生产计划文件
出库文件
临时生产计划文件
在途物资统计文件
工作令
合同文件
齐套管理
物资完成情况文件
缺料情况文件
催货文件

图 6.5 (c)

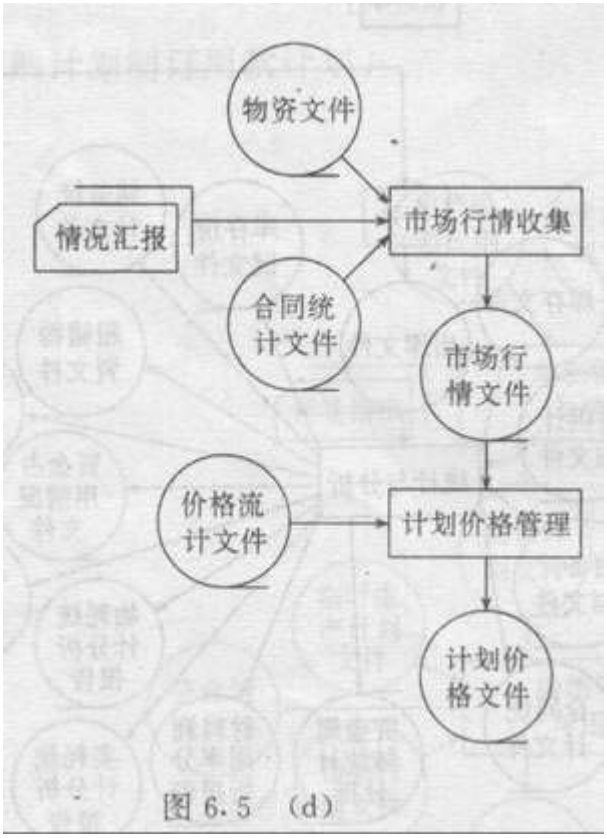


图 6.5 (d)

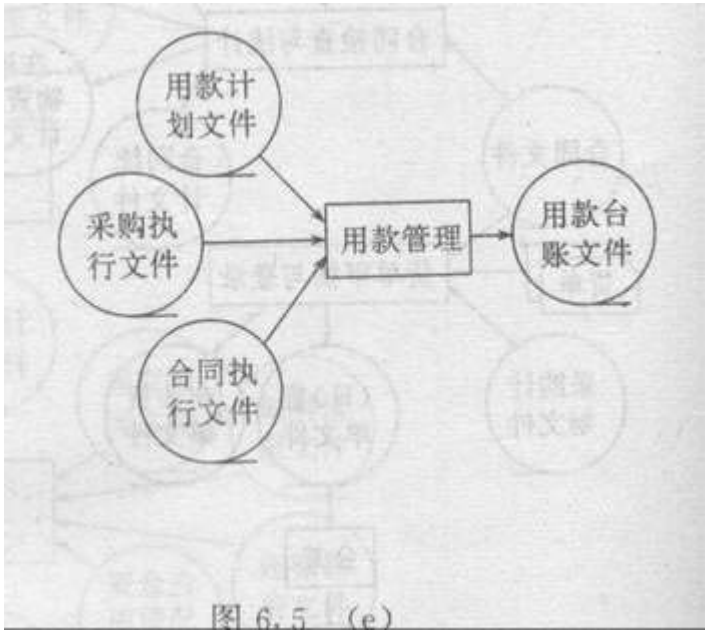


图 6.5 (e)

5．数据结构描述

物供计划文件：物资编号、型号规格、牌号、名称、技术参数、计量单位、计划单价、任务代号、全年产量、定额数、全年需用量，1-12 月份每月需用量、计划员名、物供计划编号。

物供计划通常按生产任务、分解全年产品产量对物资的需求量、并落实计划员执行。

物供计划执行文件：物供计划编号、完成量、在途量、欠交量、登记日期。

物供计划执行表明物资供应进展情况。通常合同与入库是表示进展的依据。

储备计划文件：物资编号、型合规格、牌号、名称、技术参数、计量单位、计划单价、上年消耗量、全年需用量、上半年需用量、周转储备量、计划库存量、计划员名、储备计划纺号。

储备计划执行表明物资储备进展。

采购计划文件：物资编号、（物供或储备）度划编号、计划南需用量、尚需采购量、1-12 月份计划需用量与尚需采购量、采购人员名、合同已订数、入库数、在途数、定点厂、采购计划编号。

按计划分解采购计划。

采购执行文件：采购计划编号、合同编号、合同数量、采购员名、预付款数。

使用该文件考核计划的执行比用合同文件方便，故设此文件。

使用该文件考核计划的执行比用合同文件方便，故设此文件。

用款计划文件：物资编号、采购计划编号、计划用款数、实际用款数、1-12 月份计划用款数。

按采购计划制订用款计划。

用款台账文件：物资编号、合同编号、用款资金（合计、预付款、托收付款、支票付款、凭证编号）、发票号码、经手人、备注。

跟踪用款情形。

合同或合同文件：合同编号、物资编号、型号规格、牌号、名称、技术参数、计量单位、单价、原合同号码、签约日期、供方名称、供方代表、供方地址、供方账户、邮政、电挂、电传、电话、计划员名、合同金额、订货量及进度（合计：1-12 月各月交货数）、更改标志、交货标志、运输方式、结算方式；更改后型号规格、更改后单价、更改后订货量及进度（合计：1-12 月各月交货数）、备注。

6．登记合同的基本要素

合同执行文件：合同编号、货单编号、发货量、实收量、入库量、不合格品量、检验单号、登记日期。

跟踪合同执行情况。

（日）货单文件与货单文件：货单编号、日期、物资编号、型号规格、单价、数量、合同编号、合同数、托运方式、托运日期、，发货人、备注、原货单号码。登录货单，没有合同的货单通常符合采购计划，则在备注中说明。

（日）入库文件与入库文件：日期、物资编号、型号规格、单价、入库数量、合同编号、检验单号。

记录入库物资，若干数据元素同货单文件。

库存文件：物资编号、型号规格、牌号、名称、技术参数、计量单位、计划单位、任务代号、库存量、最高库存量、最低库存量、有效期、仓位。

记录物资在仓库中数量、使用目的、位置、有效期等要素。

（日）入库文件与出库文件夹：日期、物资编号、型号规格、出库数量、计划单价、任务代号、领用单位、领用代号、领用人。

表明物资流向，。便于统计物资消耗、资金流动竺情况。

已完成合同副本文件。该文件为合同执行完毕后，将合同文件与合同执行文件联合而成的一个文件，以备查。

不合格品文件：货单编号、合同编号、数量、不合格品量、检验单号、不合格摘要、登记日期。

记录不合格品的文件。

发料单：物资编号、型号规格、名称、技术参数、计量单位、单价、任务代号。

为物资历出库凭证，有时也作进入加工线的标签。

还有一些统计、分析文件、大多与物资有关，在此不再述。

6.3.2 事务处理流程分析

事务处理流程图设计问题可分为三大类：①流程图中"处理"的设置；②信息（数据）变换与匹配问题；③数据结构问题。

1． 商场进销存调事务处理流程分析

商业企业事务处理主要集中在商品的进销存调管理和财务管理方面。其中商品进销存调的业务量与处是速度决定了企业效益。为此对进销存调事务处理流程式进行分析。

（1）"处理"的设置

在研究进销存调事务中包含四块不同的职能范围：进（商品进店）、销（商品销售）、存（商品库存）、调（商品调拔）。匹配好这四块关系；"进"要能满足"销" 的南需要，又要力所能及保证效益；"销"要加速流通，及时，准确汇总分析，提供高层决策，为"进"提供依据，同时由于自动收银机与信用卡的普遍使用，流程式也为之一新；"存"要及时，准确反映库存情况供"进"、"销"功能的实现。

图 6.6（a）为商场事务处理"进"流程图。其中包括 9 个处理结点。它是以"编制计划"、"合同管理"、"进货审核"、"验收"、"入库"为流程的主线。辅以"核价"、"调价"、"合同统计"与"异常处理"。

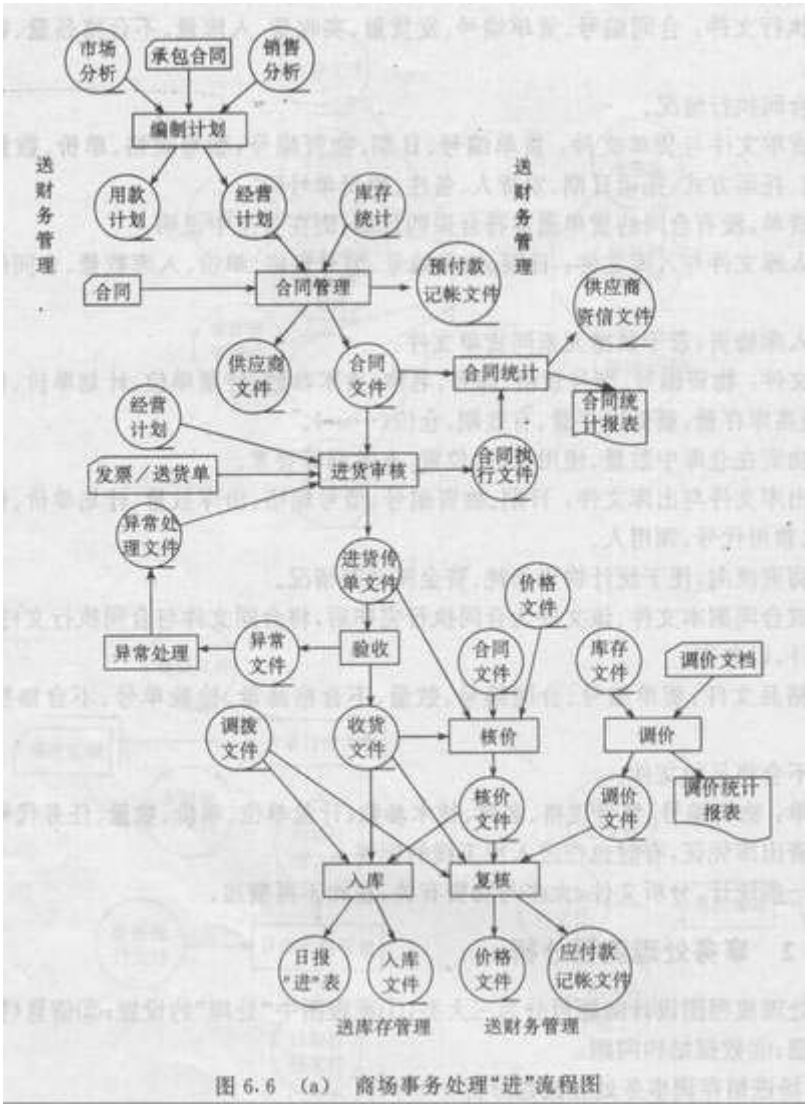


图 6.6 （a） 商场事务处理"进"流程图

分析流程中处理结点关系：商业活动完全是一种市场行为，但是它的活动却需要严格的计划。商场进货，进什么商品，进多少需要周密的计划、准确运作。因此"进"流程中以编制计划开始，编制计划与合同管理是前后关联的两个处理结点。合同是执行计划的结果，任何合同都应符合的体现，因此需要审核。但是没有合同的进货也常常发生，于是流程处理应保持一定灵活性。进货审核与验收后商品

就可入库。商品是有价格的，商品的价格经常会有变化，因此流程中应安排核价，它应在进货审核与验收之后；对核定价格还需复核，这两个结点就属流程图中服务性结点。另外，验收之后会有各种差错（包括数量、质量、规格等，此称作"异常"，为此应有异常处理结点。"调价"不属"进"流程，但与价格有关，故将这部分附于该流程之内，"合同统计"也作为"进"流程中服务结点。上述 9 个处理结点反映了"进"流程，但其中有若干处都有灵活性。

图 6.6（b）为商场事务处理"销"流程图，其中有前后台之分，前后 POS 处理流程，后台销售管理部分。前台有 6 个处理结点，后台 5 个处理结点。前后侧重于一笔交易的处理，后台侧重于销售的管理工作。

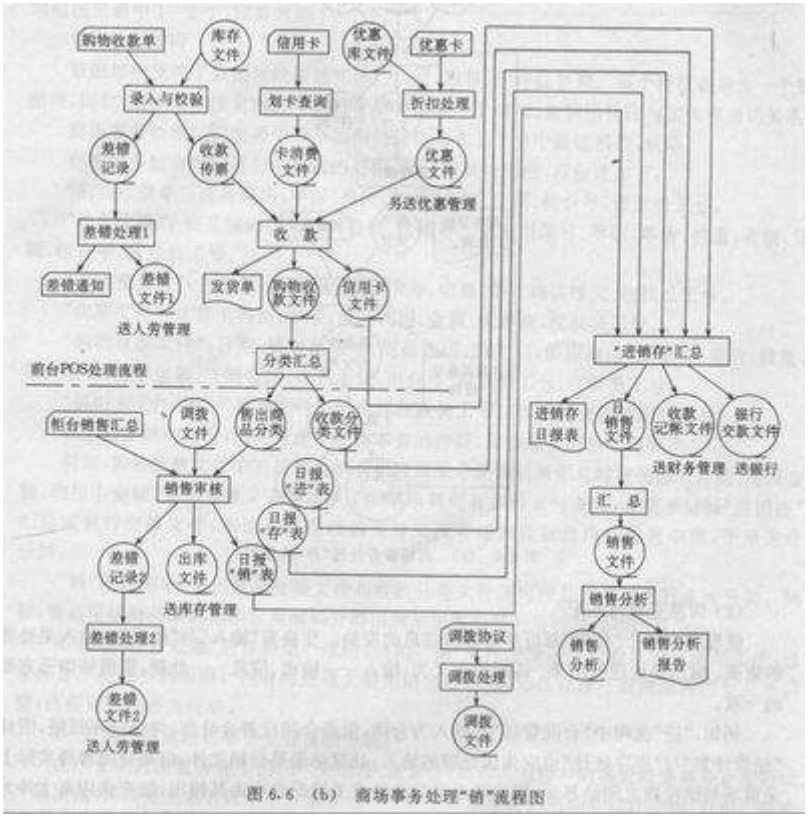


图 6.6 （b） 商场事务处理"销"流程图

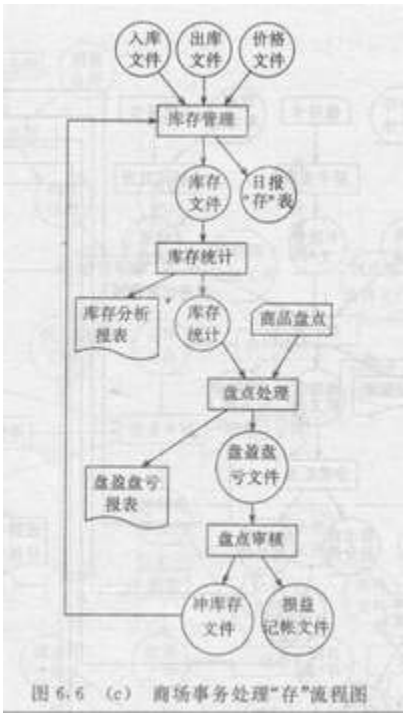


图 6.6 (c) 商场事务处理"存"流程图

前台处理以收款为中心，顾客购物交款的凭证应录入并校舍验凭证之正确性（商品代码、单价、总价和商品库存能否供应等），由于顾客可以凭证信用卡消费，因此有划卡查询工作，商场为促销发行各种优惠卡，在收款前都有应进行相应处理。销售前后台可以联机实时，也可以批处理，但前台可对售出商品与收款分类汇总。后台处理以审核 、汇总、分析为主的管理工作，其关系较简单从略。

（2） 信息变换与匹配流

流程图中每个"处理"都可看成是对信息的变换。变换有"输入"与"输出"。输入是处理的依据。输出是处理的结果。匹配的含义为：输入--输出、信息--处理、数据结构三方面的一致。

例如，"进"流程中"合同管理"的输入为合同，但是合同应符合计划，并保证不超限，因此"经营计划"与"库存统计"也应为该处理的输入；处理结果是合同文件。但是合同管理实际上是商场与供应商之间的界面，因此关于供应商的有关情况孔应由其输出，故有供应商文件为该处理的输出。又如，"进货审核"处理依据是"发票或送货单"，能否接受这批货应看是否有合同或是否与经营计划相符，某上结商品在货源充足时，旆往不预先签约，随需随送某依据只能是经营计划。进货审核的输出，表示合法的进货有"进货传单文件"，以及表示合同已执行的"合同执行文件"。其它输入一输出也是如此，通过处理的变换，互相匹配。

关于"信息-处理"的匹配，例如"销" 流程中"录入与校验处理"，不能仅仅录入与检验"购透露收款单"，如果柜台营业员单子开错，则将影响后续全部处理，故再引入"库存文件"，以便进行较全面校验。又如"收款"处理显然凭收款传票但顾客有现金消费与信用卡消费，虽然三代 POS 上有划卡装置。构成顾客与银行界面，但商场应记下信用卡消费的有关数据，还应与银行结算，故收款处理还应与"卡

消费文件"和"银行确认报文"匹配。而其输出有给顾客的"发货单"，为后续处理用的"购物收款文件"，以主为与银行结算的"信用卡文件"。

因此，在研究或回答流程图问题时，应抓住流程图中的因果关系进行分析。"处理"有目的为 "因"，处理又有"结果"与"依据"为"果"，问题总是"目的"、"依据"、"结果"三者的组合，即给出三者中 1-2 个，回答另外 1-2 个问题。

（3）数据结构

数据结构反映了数据所映射的实体（人、财、物等）的特征要素。每个特征表示为一个数据项。因此，某个实体映射出的一组特征要素形成一个结构。数据结构应与实体有密切的关系。
商场事务处理"进"流程中涉及数据的结构与 6.3.1 节中叙述相似，从略。

商场事务处理"销"流程中数据的结构，部分有其特殊性，现叙述如下：

"购物收款单"：商品编码、单位、单价、数量、金额、日期、柜台号、营业员工号。"收款传票"：商品编码、销类别（经销、代销等）、供应商编号、单位、单价、数量、金额、日期、柜台号、营业员工号。

"卡消费文件"（支票）：信用卡类别、卡号、金额、银行确认报文、收款员工号。

"优惠文件"：优惠卡类别、卡号、商品编码、金额、优惠额、收款员工号。

"购物收款文件"：日期、销售记录号、商品编码、销类别、供应商编码、单位、单价、数量、现金额、信用卡金额、支票金额、优惠金额、拒台号、营业员工号、收款员工号。"信用卡文件"同"卡消费文件"，但去掉收款员工号。作银行交款的附件使用。

以上数据结构的设计，既考虑了实体本身的特征，又参照了前后处理要求。

例如，购物收款文件中安排销类别与供应商编号是为后面汇总方便而设计的，而现金额、信用卡金额、支票金额又完全反映了一笔销售可有多种支付方式，并便于后面"进销存"汇总成银行交款文件，而这些数据项的安排又没有增加收款叫操作的复杂度，于是为合适的。

"销"流程图中，售出商品分类文件和收款分类文件夹还事按处理要求进行多种分类。例如，商品可按商品编码或供应商编码 分类心满足销售审核扔各种处理功能要求。

数据结构与"处理"的目的是一致的，例如，"销"流程中录入与校验处理与库文件，通常库存不足就不能销售。但有时处理上要先销售后入库，那么处理与数据结构均要有所调整，这些留给读者去完成。

2. 关重件质量跟踪事务处理流程分析

生产企业的质量管理中，关于"关键件、重要件"的质量跟踪事务直接影响产品质量，因而是质量管理中十分重要的一个事务。该事务的周期从设计开始直到产品检验入库为止。这个事务联系设计、生产和质量管理。为跟踪关重件质量，该事务应有三个方面目标：

（1）在设计阶段确定关重件及其质量保证要求；

（2）在生产阶段监督关重件生产的质量，采集质量信息；

（3）对关重件质量信息进行管理，以控制关重件的质量进展。

根据图 6.7 关重件质量跟踪流程图，分为下述四大类问题。

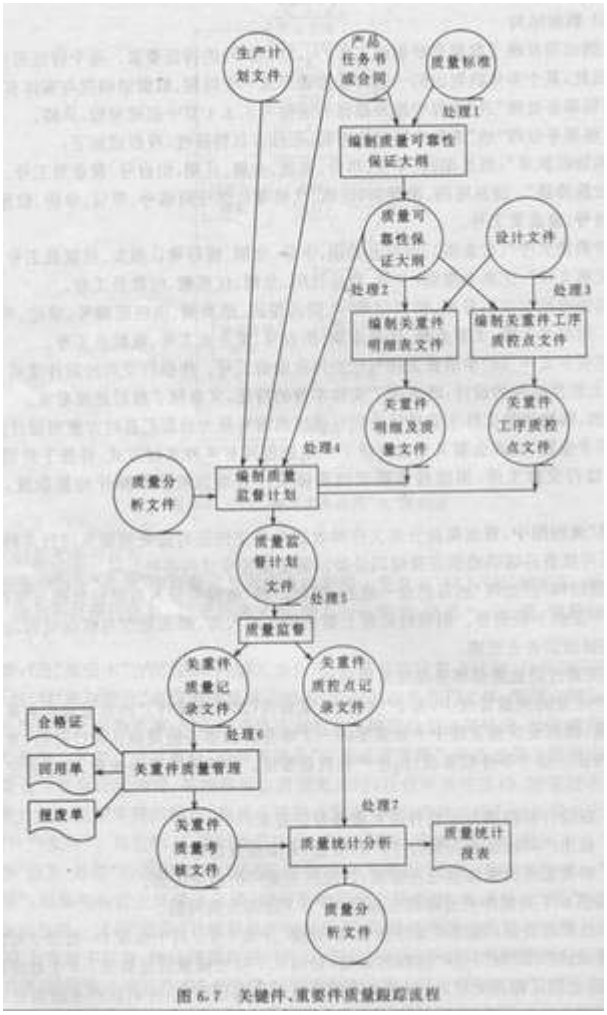


图 6.7 关键件、重要件质量跟踪流程

（1） 处理的设置：跟踪关重件质量的"处理"分成 7 个。其中处理 1-3 对应设计阶段的处理；处理 4、5 对应生产阶段的处理；处理 6、7 对应质量信息管理。7 个处理覆盖了三方面目标，它们互相是关联的，每个处理都与前后处理相联系，并可从产生的信息判定处理的功能。例如，处理 5 为质量监督，对其前驱处理应给这处理以计划，故意处理 4 为编制质量监督计划；而其后继处理，根据监督的关重件本身质量及质控点上记录这两种信息，对应两个后继处理。因此，根据处

理 5 要求，可以推出处理 4、6、7 的要求。同理，处理 4 要求是编制质量监督计划。要进行这一处理应了解：哪些是"关重件"，它们的质量要求，它们在整个产品中的保证作用，以及为保证质量在其生龙活虎工序中设置了哪些质量 控制点，根据这些才能正确地编制质量的监督计划，因而其前驱处理应有 1、2、3（当然处理 1、2、3 还有自身的关系）。

以上说明如何分析各处理间的关系如何说明一个处理的目标或要求。

另一方面，可以要求细化它们。这取决于各处理工的目标及产生的信息。例如对于设计阶段的处理，目标是"确定关重件"及其"质量保证要求"。信息有：质量可靠性保证大纲、关重件明细及质量、关重件工序质控点。后两类信息是不同类型的，因此应分别处理，它覆盖了目标中前一个的全部及后一个的部分。对于处理 2、3 的处理，应在有产品质量可靠性保证要求的前提下进行，所以必须有处理 1。故对设计阶段处理应细化为三个子处理。

（2） 信息变换与匹配：信息变换与匹配的问题，主要表现处理依据什么信息或产生什么信息。例如，图 6.7 中处理 1 产生"质量可靠性保证大纲"，它依据应有两个：①产品任务书或合同，因为它规定了产品的质量要求；②质量标准，这是国家或企业制定的法规，理应遵守。即处理 1 有两个输入；而处理 2，给出关重件明细与设计有关，因此"设计文件"应为其输入。依据的信息，可从产生的信息与处理目标相联系。而产生的信息，则按处理目标决定，有时也因下一处理要求而设置。如前所述，处理事产生的两类信息是与处理 5 的目标相联系的。

（3） 数据结构：数据结构的设计，既要研究实体本身的特征要素，又要参照其前后处理的要示。

例如，关重件明细与质量文件，它联系的是"关重件"这样的实体。这个实体应有：一名称、型号规格、技术参数、适用条件、允许误差、对应图号、或者还有引用标准等要素。将这些要素组合起来成为一个结构，然后再参照前后处理要求，分析还应具备的数据项。例如，为了制定质量监督计划，要参考核关重件试验时间，并在明细的数据结构中列这一时间。经过这种与前后联系的加工，以便完善一个数据结构。

又如，关重件工序质控点文件，它联系的是质量控制点这样的实体。这个[实体应有：一个位置、联系的关重件名、联系的工序号、联系的工艺号、有关技术参数、控制参数、允许误差、操作人员名等要素。数据结构应包含它们，并且在数据结构中还应联系"关重件质控点记录文件"。它们之间的数据项有相应匹配关系，将这些加入质控点文件的数据结构中去，成为一个完整的结构。

习题

6.1 就 6.3.1 节中物资供应流程设计而言，如果有下述要求，如何修正流程：

（1） 为了减少库存，压缩资金占用，某些材料要求所谓"零库存"应在流程图何处进行修正？

（2）库存管理中若要增加超限处理，为了控制采购与合同的执行，应作何修正？

（3）在限额发料中，若领用单所述物资缺乏，是否采用代用，流程应作何修正？

同时就该例，考虑如下问题：

（4） 为什么引入采购执行文件与合同执行文件？

（5） 为什么货单审核与登录使用采购计划文件？

6.2 设备使用管理的流程图设计

设备使用管理为设备管理的后期工作，这时已有设备台账。设备使用管理的目标为：

（1） 根据生产需要，对设备进行保养（一、二级保养这里不作细分）。

（2） 根据生产需要与设备性能，安排设备检测（年检或月检等）。

（3） 根据故障、保养、检测，安排设备大修理。

（4） 对设备故障、事故进行处理。

（5） 对设备进行统计分析。

根据以上目标设计事务处理流程图，其中要求保养、检测、大修理均应有计划。

6.3 工厂成本核算这一事务，假设采用定额法进行成本核算，则工厂成本的成本项目包括：（1）原材料费用；（2）燃料和动力费用；（3）工资；（4）提取的职工福利基金（利润留成企业无此项）；（5）废品损失；（6）车间经费；（7）企业管理费。

若用定额法进行成本核算，请指出（1）、（3）两项的数据结构。

6.4 客车调度事务流程如图 6.8，请按本书采用的形式细化该程图。

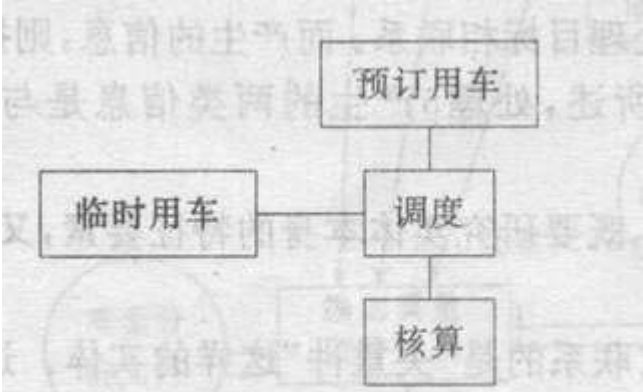


图 6.8 客车调度事务流程图

6.5 设计一个经销流程图。其要求包括销售计划管理、合同管理、发货管理、成品库管理、财务结算、统计分析。

销售计划管理：按市场需求、工厂生产能力制订销售计划，并检查计划的执行。

合同管理：根据计划落实销售合同，及时执行合同。

发货管理：按合同及库及时组织运输计划及发货。

成品库理：有效管理产品入库、出库及存放。

财务结算：按合同预收款或托收，出具发票、通知发货、并进行财务结算。

统计分析：销售、库存、合同等统计分析。

请画出其流程图及数据词典。

第 7 章 CASL 程序设计

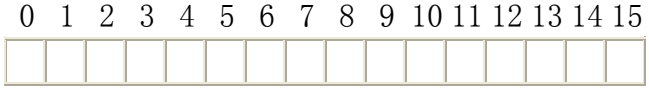
7.1 CASL 概述

水平考试中采用的 CASL 汇编语言是 COMET 模型机上的汇编语言。与其它汇编语言相比，它更加简洁精巧，便于阅卷评分，适用于考试，且具有足够强的功能。CASL 由 4 种指令、3 种宏指令和 23 种符号指令组成。

7.1.1COMET 机的硬件结构

1. COMET 机是一台字长为 16 位的定点计算机，主存储器的容量是 65536 字，按编号 0000-FFFF（十六进制）编址。

2. 一个字的 16 位二进制位采用自左至右的次序编号，即



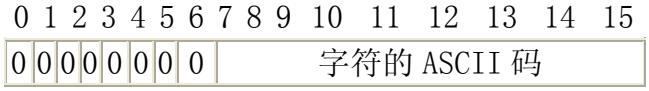
3. 一个字中的 16 位可以是：

(1)不带符号的二进制非负整数。此时一个字应能表示的数的范围是：$0 \leqslant x \leqslant 2^{16}-1$

(2)用补码表示的带符号的二进制数。此时一个字能表示的数的范围是：
$-2^{15} \leqslant x \leqslant 2^{15}-1$。

（3）地址常数。此时一个字能表示的地址写成十六进制时是：0000-FFFF。

（4）字符数据。此时一个字的高八位皆应为零，低八位为字符的 ASCII 码，即



4. COMET 机具有 5 个通用寄存器 GR（16 位）、一个指令计数器 PC（16 位）和一个标志寄存器 FR（2 位）。它们的作用如下所述：

GR（通用寄存器）：5 个通用寄存器的编号为 0、1、2、3、4 分别记为 GR0、GR1、GR2、GR3、GR4。这些通用寄存器用于算术、逻辑、比较、移位等运算。其中 GR1、GR2、GR3、GR4 通用寄存器兼作变址器，GR0 通用寄存器不能兼作变址器。另外，GR4 还兼作栈指针（SP），栈指针是存放顶地址用的寄存器。

PC（指令计数器）：在指令执行过程中，PC 中存放着正在执行的指令的第一个字的地址（一条指令占两个字）。当指令执行结束时，置入下一条将要被执行的指令的第一个字的地址。也就是说在一条指令执行结束时，一般把 PC 的内容加 2，只有在执行转移指令且转移条件成立时或执行调用子程序指令时，才将转移或调用地址置入 PC 中。

FR （标志寄存器）：在 LEA、ADD、SUB、AND、OR、EOR、CPA、CPL、SLA、SRA、SLL、SRL 等指令执行结束时，根据指令执行的结果，将 FR 置成 00、01 或 10。其它指令的执行不改变 FR 的值。

5. COMET 的控制方式为顺序控制，指令的 32 位二进制位构成，即双字长，其结构没有定义。

6. 指令的符号表示：

GR 泛指通用寄存器，可用数字 0、1、2、3、4 来指定，也可以用 GR0、GR1、GR2、GR3、GR4 来指定。

XR 泛指变址寄存器，可用数字 1、2、3、4 来指定，也可以用 GR1、GR2、GR3、GR4 来指定。

SP 栈指针（第 4 号通用寄存器）。

ADR 为标号（表示与标号对应的地址），为十进制数（-32768≤ADR≤65535；ADR 作为地址具有 0-65535 的值，而 32768-65535 的值也可以用负的十进制数表示）。

E 表示有效地址。形成有效地址的规则是：E=ADR[+（XP）]（mod216）
[ ]表示被[ ]包围的部分可以省略（以下同）。XP 省略时，表示不使用变址寄存器。

7.1.2 指令系统

1. 指令的种类和书写格式

CASL 由 4 种伪指令（START、END、DS、DC）、3 种宏指令（IN、OUT、EXIT）和 23 种符号指令（COMET 的指令）组成。指令供汇编者按系统确定汇编程序的结构，定义汇编程序中使用的常数、变量和数据区；宏指令的功能由操作系统实现；符号指令是 COMET 机执行的指令。

CASL 的每条指令书写在一行内（最多不超过 72 个字符），指令的书写格式如表 7.1 所示。

表 7.1 CASL 指令书写格式

标 号	指 令 码	操 作 数	注释
[LABEL]	START	[LABEL]	
空 白	END	空 白	
[LABEL]	DC	常 数	
[LABEL]	DS	区域的字数	
[LABEL]	IN	ALABEL,NLABEL	
[LABEL]	OUT	ALANEL,NLABEL	
[LABEL]	EXIT	空 白	
[LABEL]	符号指令参照下面的 4		

由表 7.1 可知，CASL 的每条指令由标号（可缺省）、指令码、操作数（可缺省）和注释（可缺省）共 4 栏组成。每一栏的收写规则如下：

标号栏：从第一个字符开始，最多不超过 6 个字符位置。

指令码栏：无标号时从第二个字符位置以后的任意字符位置开始。

操作数栏：指令码后至少有一个空白，其后到第 72 个字符位置 。不能继续到下一行。

注释栏：行里有分号（；）时，其后直到终了作为注释处理（但 DC 指令里的字符串中的分号除外）。此外，在第一个字符位置为分号或在分号前只有空白的情况下，该行全部作为注释处理。在注释栏里可以书写任保字符。

LABEL 泛指标号。标号最多不超过 6 个字符，开头必须是英文大写字母，以后可为英文大写字母或数字。

用空白表示的栏目里不得写入字符。

2. 伪指令

（1）[LABEL] START [LABEL]

表示程序的开头，即在程序开始必须书写。

操作数栏中的标号是这个程序中定义的标号，它指出该程序的启动地址。在省略的情况下，程序从 START 下面的一条指令开始执行。

标号栏中的标号可以作为其它程序进入该程序的入口，它供其它程序调用该 程序时使用。

（2）END

表示程序的终止，在程序的末尾必须书写。每个程序都以 START 开始，以 END 结束。

（3）[LABEL]DC 常数

用来指定和存储常数。常数分十进制常数、十六进制常数、地址常数和字符串常数共四种。

标号栏中的标号是代表被指定的十进制常数、十六进制常数、地址常数的存储地址或代表被指定的字符串常数的存储区域的第一个字的地址。

十进制常数：DC n

用 n 来指定一个十进制数（-32768≤n≤65535），并将 n 转换成二进制数存储在一个字中。如果 n 超出规定的范围，则将其低 16 位存储起来。

对于 32768-65535 的十进制数也可以用负的十进制数表示。

十六进制数：DC # h

用 h 指定一个 4 位十六进制数（0000≤h≤FFFF），并将 h 对应的二进制数存储在一个字中（在 h 的前面必须写上#）。十六进制数的数字字符为 0-9 和 A-F。

地址常数：DC LABEL

将标号 LABEL 所对应的地址作为一个字的二进制数存储。若 LABEL 在该程序中没有定义，汇编将保留地址的定义，并由操作系统处理。

字符串常数：DC ’字符串’

将字符串中从左开始的每个字符转换成字符数据，并依次把字符数据存储在连续的各字中。

在字符串中，不允许出现字符 ’；字符串的长度不能是零（即空字符串）。

例如：伪指令 DC ’ WANG’S ’ 和 DC ’ ’均为错。

（4）[LABEL]DS 区域的字数

用来保留指定的字数的存储区域。

区域的字数用十进制数（≥0）指令。

标号栏中的标号代表被保留的存储区域的第一个字的地址。

区域的字数为零时，存储区域不存在，但是标号栏中的标号仍有效，即代表下一指令所在字的地址。

例如：

标号	指令码	操作数	说明
WORK	DS	1	保留一个字的存储区域
INBUF	DS	80	保留 80 个字的存储区域
SAVE	DS	1	保留一个字的存储区域
STACK	DS	0	未保留存储字,STACK 表示下一指令所
	DC	#FFFF	在字的地址
			定义十进制常数

3. 宏指令

宏指令是根据事先定义的指令串和操作信息，生成指定功能的指令串。

CASL 中有进行输入、输出及结束和程序等宏指令，而没有定义输入输出符号指令，这类处理由操作系统完成。

程序中出现宏指令时，CASL 生成调用操作系经的指令串，但生成的指令串的字数不定。

执行宏指令时，GR 的内容保持不变，而 FR 的内容不确定。

（1）[LABEL] IN ALABEL，NLABEL

宏指令 IN 从输入装置上输入一个记录，记录中的信息（字符）依次按字符数据的形式顺序存放在 ALABEL 开始的区域内，已输入的字符个数以二进制形式存放在标号为 NLA-BEL 的字中。记录之间的分隔符不输入。

（2）[LABEL] OUT ALABEL，NLABEL

宏指令 OUT，将存放在从标号 ALABE 开始的区域中的字符数据，作为一个记录向输出装置输出，输出的字符个数由标号为 NLABEL 的字中的内容指定。输出时，若要记录间的间隔符号，由操作系统自动插入输出。

CASL 只支持字符型输入输出设备。

例如：

标号	指令码	操作数	说明
	IN	BUF，L	从中端输入设备输入一个记录
	OUT	BUF，L	将输入的记录在输出设备上输出
BUF	DS	80	保留从 BUF 开始的 80 个存储区域
L	DS	1	保留一个字的存储区域

（3）[LABEL] EXIT

宏指令 EXIT，表示程序执行的终止，控制返回操作系统。

4. 符号指令

CASL 中有 23 种符号指令，按其功能分为八类。

（1）存取数指令

名称	指令码	操作数	功能
取数	LD	GR，AND[，XR]	（E）→GR
存数	ST	GR，AND[，XR]	（GR）→E

（2）取地址指令

名称	指令码	操作数	功能
取地址	LEA	GR，AND[，XR]	E→GR

*该指令还根据执行结果（GR 中的值），形成 FR 的值，其规则是：

若 GR 中的第 0 位为 0 且其余各位不全为 0，则 FR 被置成 00；

若 GR 中的 16 位全为 0，则 FR 被置成 01；若 GR 中的第 0 位为 1，则 FR 被置成 10。

后面的加减运算指令、逻辑运算指令和移位指令也根据执行的结果设置 FR 的值，其规则与此相同（后面注*的指令）。

LEA 指令和 LD 指令不同：①LEA 指令是将有效地址 E 送 GR ，而 LD 指令是将有效地址 E 所指单元的内容送 GR；②LEA 指令根据执行结果设置 FR ，而 LD 指令不影响 FR 。

LEA 指令不依赖于内存单元的内容，且该指令的执行结果又设置标志寄存器 FR，在程序设计中该指令使用较为广泛。该指令在程序中的作用可以归纳以下几点：

①给寄存器设置初值：在循环结构的初始化部分中，常用该指令设置初值。

②调整寄存器的值：在循环结构的调整部分，常用该指令调整 GR1-GR4 中某一寄存器的值。

③测试寄存器的值：

指令：LEA GR1，0，GR1

即根据 GR1 的值设置 FR。因此利用 LEA 指令可以测试 GR1-GR4 中某一寄存器的值。在循环结构中该指令用于循环控制。程序中，在这样使用的指令后面一般都跟一个条件转移指令。

④寄存器的值加减一个常数送另一寄存器。

⑤一寄存器的值送另一寄存器。

⑥内存单元地址送寄存器。

在主子程序结构中，若主程序要将一组信息传送到子程序，则可利用该指令将存放这组信息的存储区域的首单元的地址送某一寄存器传给子程序。在链表结构中也常常使用该指令将结点的指针送寄存器。

例如（见下表）：

由于 LEA 指令在程序设计中具有上述的作用，因此是考试中的重点考核内容。

标号	指 令 码	操 作 数	说 明

	LEA	GR2,0	给 GR2 设置初值 0→GR2，设置 FR
	LEA	GR2,1,GR2	调整 GR2 的值（GR2）+1→GR2，设置
	LEA	GR3,-1,GR3	FR
	LEA	GR3,0,GR3	调整 GR3 的值（GR3）-1→GR3，设置
	LEA	GR0,5,GR2	FR
	LEA	GR1,SOUT	设置 FR
	LEA	GR1,LIST,GR2	GR2 的值加一数送 GR0,(GR2)+5→GR0,
	LEA	GR1,0,GR3	设置 FR
SOUT	LEA	4	标号 SOUT 对应地址送 GR1,SOUT→GR1,
LIST	DS	200	设置 FR
	DS		（GR2）+LIST→GR1，设置 FR
			（GR2）+LIST→GR2，设置 FR
			GR3 的值送 GR1，（GR3）→GR1，设置
			FR
			保留从 SOUT 开始的 4 个字的存储区
			域
			保留从 LIST 开始的 200 个字的存储
			区域

完成一定功能的程序，一般都含有循环结构。LEA 指令可以进行循环的初值设置、循环控制变量的调整、循环测试和循环控制。因此必须熟练掌握 LEA 指令的使用。

（3）加减运算指令

名 称	指令码	操作数	功 能
加 法	ADD	GR,ADR[,XR]	(GR)+(E)→GR,设置 FR
减 法	SUB	GR,ADR[,XR]	(GR)-(E)→GR,设置 FR

例如：下面的指令序列实现 2（X）-（Y）→GR0

标 号	指 令 码	操 作 数	说 明
	LD	GR0,X	(X)→GR0
	ADD	GR0,X	(X)+(GR0)→GR0
	SUB	GR0,Y	(GR0)-(Y)→GR0
X	DS	1	保留一个字的存储区域
Y	DS	1	保留一个字的存储区域

CASL 中没有溢出标志。两个同号数相加或两个异号数相减可能产生溢出，而在标志寄存器 FR 中没出有反映溢出的标志。因此读者在进行程序设计时要注意这一点。同样在后面将要介绍的移位指令中，如果产生溢出，在标志寄存器 FR 中也没有任何反映。

（4）逻辑指令

名 称	指令码	操作数	功 能
逻辑乘*	AND	GR，ADR[，XR]	（GR）∧（E）→GR,设置FR
逻辑加*	OR	GR，ADR[，XR]	（GR）∨（E）→GR,设置FR
按位加*	EOR	GR，ADR[，XR]	（GR）∀（E）→GR,设置FR

例如：利用 EOR 指令可以判断两个有符号数是否同号。下面的指令序列判断 X，Y 单元的最高位是否相同，结果存于 FR 。

标 号	指 令码	操 作 数	说 明
	LD	GR0,X	(X)→GR0
	EOR	GR0,Y	(GR0)∀(Y)→GR0,设置 FR
X	DS	1	保留一个字的存储区域
Y	DS	1	保留一个字的存储区域

在程序设计中，利用逻辑乘指令，可以析取寄存器或某一个字中的某几位，也可以给寄存器或某一字中的某几位置 0；利用逻辑加指令，可以给寄存器或某一字中的某几位置 1；利用按位加指令，可以给寄存器或某一字中的某几位取反等。

名 称	指令码	操作数	功 能
算术比较	CPA	GR,ADR[,XR]	视(GR)和(E)为用补码表示的带符号的二进制整数
			若(GR)>(E)则 00→FR
			若(GR)=(E)则 01→FR
			若(GR)<(E)则 10→FR
逻辑比较	CPL	GR,ADR,[,XR]	视(GR)和(E)为无符号的二进制非负整数
			若(GR)>(E)则 00→FR
			若(GR)=(E)则 01→FR
			若(GR)<(E)则 10→FR

读者要仔细体会算术比较与逻辑比较的差别;两种比较有时会得出相同的结果 ，有时会得出不同的结果。如：-32768 与-32767 分别进行算术比较和逻辑比较结果会怎样？-32768 与 32767 分别进行算术比较和逻辑比较结果又会怎样？

不出现溢出的减法运算 SUB，在设置 FR 方面和算术比较指令 CPA 一致。但减法指令还可以改变通用寄存器 GR 的值。

（6）转移指令

名 称	指令码	操作数	功 能
无条件转移	JMP	ADR[,XR]	无条件向有效地址 E 转移,即 E→PC
大于等于转移	JPZ	ADR[,XR]	FR 的值为 00 或 01 时转向 E,即 E→PC;否则(PC)+2→PC
小于转	JMI	ADR[,XR]	FR 的值为 10 时转向 E,即 E→PC;否则(PC)+2→PC
不等转	JNZ	ADR[,XR]	FR 的值为 00 或 10 时转向 E,即 E→PC;否则(PC)+2→PC
等于转	JZE	ADR[,XR]	FR 的值为 01 时转向 E,即 E→PC;否则(PC)+2→PC

与前面介绍的几类指令不同的是，转移指令的操作数中无通用寄存器 GR。转移指令是 CASL 中的一类重要指令。在分支结构、循环结构的程序中都有要用转移指令。在条件转移指令中没有大于转和小于等于转，用两条转移指令就可以分别实现大于转和小于等于转。

例如：大于转和小于等于转的实现。

名 称	标 号	指令码	操作数	说 明
大于转	A	JZE JPZ …	A ADR[,XR]	FR 的值为 10 时转 A FR 的值为 00 时转 E FR 的值为 10 或 01 时转执行该指令
小于等于转		JZE JMI	ADR[，XR] ADR[，XR]	FR 的值为 01 时转 E FR 的值为 10 时转 E

例如：下面的程序是一子程序，用来统计 GR1 所指内存单元中"1"的个数，结果存于 GR2 返回。统计"1"的个数的方法为：

a∧(a-1)去掉 a 最左边为"1"的位。

标 号	指令码	操作数	说 明
BITCNT	START		
	LEA	GR2，0	计数器设初值，0→GR2
	LD	GR3，0，GR3	取 GR1 所指的单元的内容到 GR3
	LEA	GR3，0，GR3	测试 GR3 的值
LOOP	JZE	RT	为零转 RT

	LEA	GR2，1，GR2	计数器增 1，（GR2）+1→GR2
	ST	GR3，WORK	a∧(a-1)
	LEA	GR3,-1,GR3	
	AND	GR3，WORK	
	JMP	LOOP	转 LOOP
RT	RET		返回主程序，这是一条返回指令，详见
WORK	DS	1	后面介绍
	END		保留一个字的存储区域

（7）移位指令

名 称	指令码	操作数	说 明
算术左移*	SLA	GR,ADR[,XR]	把（GR）向左或向右移动 E 位。在算术移位时，GR 中的第 0 位保持不变，在右移时空出的位被置成与第 0 位相同的 1 或 0；在左移时空出的位被置成 0。在逻辑移位时，GR 中的 16 位一起移动，而空出的位被置成 0。
算术右移*	SRA	GR,ADR[,XR]	
逻辑左移*	SLL	GR,ADR[,XR]	
逻辑右移*	SRL	GR,ADR[,XR]	

在算术左移或逻辑左移指令中有时会产生溢出，但 CASL 中没溢出标志，因此在程序设计中。使用到这类指令要注意溢出情况。在没有溢出的情况下，算术左移指令 SLA，实现（GR）*2E-GR；算术右移指令 SRA，实现 ┖(GR)/2^{E}┘ →GR。

例如：已知（GR1）=#80F5，（FR）=10，指令 SLA GR1，4 执行后，（GR1=#8F50，）（FR）=10；指令 SRA GR1，4 执行后，（GR1）=#F80F，（FR）=10；指令 SLL GR1，4 执行后，（GR1）=#0F50，（FR）=00；指令 SRL GR1，4 执行后，（GR1）=#080F，（FR）=00。

（8）堆栈指令

CASL 中的栈指针由 GR4 给出。栈区既可以由系统定义，也可以由用户军定义，如果用户在程序中未定义栈区，则使用系统栈区。如果用户在程序中定义了栈区，则使用用户栈区，系统栈区作废。就是说在用户程序中既可以定义栈区也可以不定义栈区。所谓栈区就是一个按照先进后出的原则组织的一段内存区域，并且有一个栈指针 SP 指向栈顶元素。末存放任何元素的栈称为空栈，此时栈指针 SP(GR4 兼作）指向栈底的下一个单元，如图 7.1 所示。



图 7.1 栈空时的状态

名称	指令码	操作数	说 明
进栈	PUSH	ADR[,XR]	(SP)-1→SP, E→(SP)
退栈	POP	GR	((SP))→GR, (SP)+1→SP
调用	CALL	ADR[,XR]	先形成有效地址，再依次执行(SP)-1→SP; (PC)+2→(SP); E→PC
返回	RET		((SP))→PC, (SP)+1→SP

使用 PUSH 指令可将 GR1-GR4 中任一寄存器的值压栈保存。主程序通过 CALL 调用子程序，子程序通过 RET 返回主程序，它们之间的关系如图 7.2 所示。

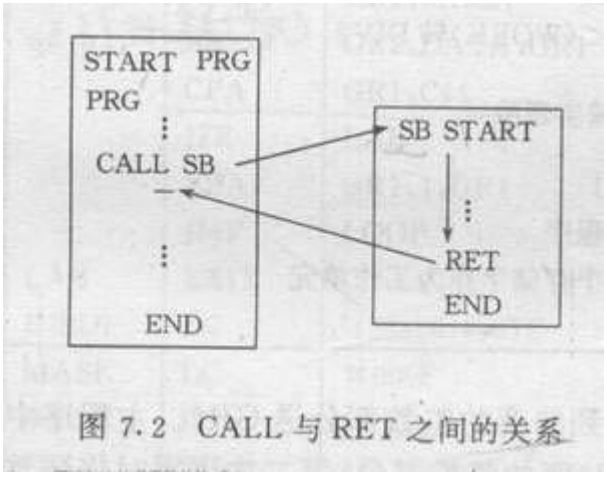


图 7.2 CALL 与 RET 之间的关系

例如：下面的程序是一个除法程序。

① 被除数、除数均是正数，分别存放在标号为 A、B 的存储字内。

② 商存储在标号 ANS 存储字内，对商的小数点后第 1 位进行四舍五入成整数。

③ 假定除法过程不会产生溢出。

本程序在除法中实现四舍五入的方法是：将被除数乘以 10，除以除数，对商作适当处理，再将处理后的结果除以 10。

[程序]

标 号	指令码	操作数	说 明
	START	BGN	
BGN	LD	GR0,A	取被除数
	SLA	GR0,3	被除数扩大 10 倍送 GR0
	ADD	GR0,A	
	ADD	GR0,A	
	LD	GR1,B	取除数
	CALL	DIV	调用除法子程序 (GP0)÷(GR1)→GR2
	LEA	GR0,5,GR2	对商加 5 调整 GR0
	LEA	GR1,10	10→GR1
	CALL	DIV	调用除法子程序 (GP0)÷10→GR2
	ST	GR2,ANS	存储结果
	EXIT		程序执行终止,控制返回操作系统
A	DS	1	保留存放被除数的一个存储字
B	DS	1	保留存放除数的一个存储字
ANS	DS	1	保留存放商的一个存储字
DIV	ST	GR1,WORK	除法子程序入口,(GR1)→WORK
	LEA	GR2,0	0→GR2
DIV1	CPA	GR0,WORK	(GR0)与(WORK)比较
	JMI	DIV2	(GR0)<(WORK)转 DIV2
	LEA	GR2,1,GR2	用连减实现除
	SUB	GR0,WORK	
	JMP	DIV1	转 DIV1
DIV2	RET		返回主程序
WORK	DS	1	保留一个存储字作为工作单元
	END		

该程序中，子程序 DIV 实现（GR0）÷（GR1）得到的商的整数部分送 GR2。主程序中两次调用子程序 DIV。第一次调用时求 10*（A）÷（B）商的整数部分，第二次调用时将调整后的商缩小 10 倍。

7.1.3 CASL 汇编简介

CASL 的汇编过程就是将 CASL 汇编源程序中的符号指令翻译或 CASL 目标代码(机器指令）。

COMET 机的机器指令的结构虽未明确定义，但不失一般性，可将符号指令的目标代码设想成如图 7.3 所示。



图 7.3 符号指令的代码结构

其中：

OP：指令码，由 8 位组成的代码，分别表示 23 种符号指令。

GR：通用寄存器，由 4 位组成的代码，分别表示 GR0-GR4，5 个通用寄存器。

XR：变址寄存器，由 4 位组成的代码，分别表示 GR0-GR4，5 个变址寄存器。

ADR:形式地址,由 16 位组成,具有 0-65535 的值.

将 CASL 源程序汇编成目标代码，一般采用两遍扫描的方法。第一遍扫描进行伪指令处理、确定各标号的地址以及给一条指令分配地址。第二遍扫描形成指令目标代码。

下面结合一简单的 CASL 程序，说明汇编过程的几个要点。

例如：下面的程序是将 INBUF 开始的存储区域中的 12 个字符转换成 12 个整数存于从 DATA 开始的存储区域中的各单元。该程序执行后，从 DATA 开始的 12 个单元的值分别为 1、2、3、4、5、6、7、8、9、0、1、2。

标号	指令码	操作数	说明
LOOP	START LRA LR AND ST CPA JZE LEA JMP	 GR1,0 GR2,INBUF,GR1 GR2,MASK GR2,DATA,GR1 GR1,C11 LAB GR1,1,GR1 LOOP	 0→GR1,INBUF 存储区相对指针置初值 取 INBUF 存储区中 GR1R 中所指单元的内容到 GR2 取 GR2 的低 4 位，（GR2） ∧#000F→CR2 将（GR2）送 DATA 存储区中 GR1 所指单元

整理人：刘海云
整理时间：2003.05

LAB	EXIT		（GR1）与此同时 11 比较
INBUF	DC	’1223456789012’	（GR1）=11 转结束
MASK	DC	#00F	调整存储区相对指针，（GR1）
DATA	DS	12	+1→GR1
C11	DC	11	转 LOOP
	END		程序终止执行，控制返回操作系统
			定义从 INBUF 单元开始的字符串常数
			定义十六进制常数 00F
			保留从 DATA 开始的 12 个字的存储区域
			定义十进制常数 11

1、DC,DS 伪指令处理

该程序从 START 开始到 END 结束。汇编过程中，给 START 下面第一条指令 LEAGR1，0 分配相对地址为 0,1 的单元，由于一条指令占两个字，因此第二条指令分配在相对地址 2、3 的单元，如此下去。当汇编到指令 INBUF DC ’123456789012’ 时，发现是一个定义字符串常数的伪指令，则将’ ’内的每个字符各装入一个单元，存放字符的单元的高八位全为 0，低八位为字符的 ASCII 码。汇编 MASK DC #000F 伪指令时，将十六进制常数 000F 以二进制形式装入一单元。汇编 DATA DS 12 伪指令时，保留 12 个连续单元。如此下去，一直遇到 END 为止。该程序的相对地址空间如图 7。4 所示。

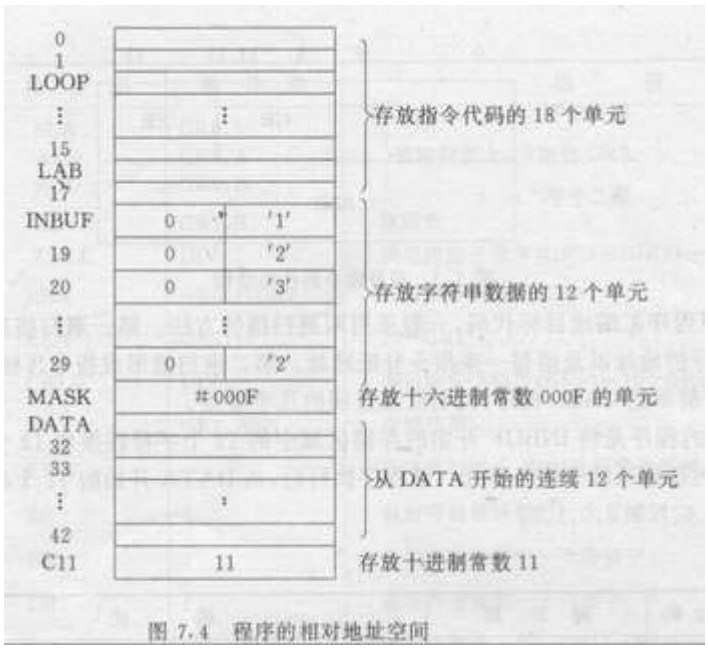


图 7.4 程序的相对地址空间

2、标号处理

每个 CASL 程序一般都含有标号，标号实际上是符号地址。在程序的标号栏位置上出现的标号，称为对标号的定义。它定义了标号值（地址）。在该程序中共定义了 6 个标号，它们所对应的地址如表 7.2 所示.

表 7.2 该程序中标号对应的地址

标 号	对应的相对地址	标 号	对应的相对地址
LOOP	2	LAB	16
LNBUF	18	NASK	30
DATA	31	C11	43

在操作数栏位置上出现的标号，称为对标号的引用。如，指令 AND GR2，MASK 的操作数中的标号 MASK 就表示相对地址为 30 的单元（也称 MASK 单元）。只有在标号栏出现了的标号在操作数栏才能引用。

CASL 中规定，标号由大写字母和数字组成，并且由大写字母打头，组成标号的字符个数不能超过 6 个。

如：L1,L1A,C01,CFF 都是标号。

1L（由数字打头）、ABBBBBB（太长，7 个字符）、A6q（出现大写字母和数字以外的字符）都是不合法的标号。

3、CASL 程序中的常见错误

编写程序时出现一些错误常常是不可避免的。但初学者在进行程序设计时要注意以下两点。

（1） 不要将 GR0 作变址器使用，如

标号	指令码	操作数
	LEA	GR0，1，GR0
	ST	GR1，BUF，GR0
	ADD	GR2,0,GR0
	PUSH	0,GR0

这些指令都不对，因为它们将 GR0 作变址器使用了。

（2） 注意 LD 和 LEA 的区别

LD 指令的功能是将有效地址 E 所指单元的内容送到 GR，即（E）→GR：而 LEA 指令的功能是将有效地址 E 送到 GR，即 E-GR，且设置 FR。

如，指令：LD GR0， 0， GR1

是将 GR1 所指单元的值送 GR0，即（（GR1））→GR0。

指令：LEA GR0, 0, GR1

是将 GR1 的值送到 GR0，即：（GR1）→GR0，且设置 FR。

如，指令：LD GR0， 0

是将 0 号单元的值送 GR0，即：（0）→GR0

而指令：LEA GR0，0

是将 0 送 GR0，即：0→GR0，且设置 FR 为 01。

7.2 CASL 程序设计

本节介绍 CASL 汇编程序中三种常用程序结构（分支、循环、子程序调用）的设计特点和方法，并举例介绍频繁使用的字符串的处理和输入/输出的程序设计方法。

7．2．1 分支

分支结构是一种基本程序结构,完成一定功能的程序一般都会有一个或多个分支结构。构成分支结构的关键是条件转移指令。条件转移指令是根据 FR 的值进行转移的，因此在条件转移指令前都必须预先设置 FR 的值。FR 的值是执行加减运算指令、逻辑运算指令、移位指令、比较指令和取地址指令后，根据其结果设定的。分支结构一般可分为二分支结构（如图 7.5a 所示）和多分支结构.多分支结构可以是由多个二分支结构构成的,也可以是通过分情况转移的方式构成的(如图 7.5(b).(c)所示).

图 7.5(a)所示的二分支结构中,一般可采用一条件转移指令实现二分支.图 7.5(b)所示的多个二分支组成的多分支结构中,则要采用多个条件转移指令实现多分支.图 7.5(c)所示的多分支结构,则先要定义一张与各种情况对应的转移地址表,然后在程序中求出情况值,按不同情况查到相应地址后再进行转移。

[例 7.1] 求函数值.函数 y 定义如下：

y=

a+b-c　　d=2 时
a-b-c　　d=3 时
a∧b-c　　d=4 时
a/∧b-c　　d=5 时
a∀b-c　　d=其它时

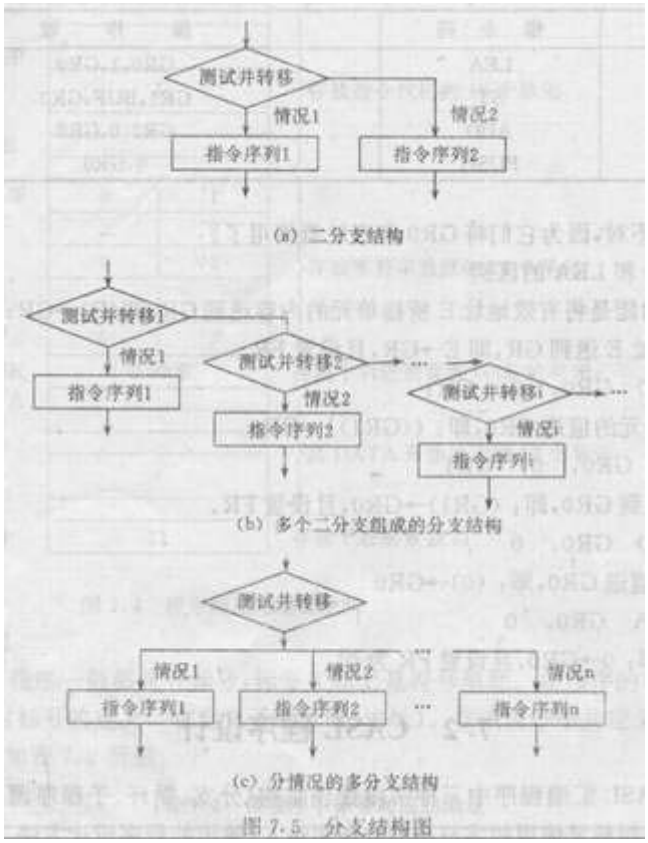


图 7.5 分支结构图

下面一子程序用于求 y 值,并将结果存于 GR0 返回.调用子程序时 a,b,c,d,存于 A,B,C,D 单元.

[问题]填充下面程序中的空格,完成程序,不必要的标号不要写.

[程序]

标号	指令码	操作数	说明
SB	START		
	PUSH	0，GR1	保护寄存器
	PUSH	0，GR2	
	LD	GR0，A	
	LD	GR1，D	
		{ ① }	
	JMI	DSOR	
	CPA	GR1，C4	
	JPZ	DXOR	
		GR2，	
	LD	ENTRY，	
		GR1	
		{ ② }	
	EOR	GR0，B	
		{ ③ }	

```
        POP     GR2
        POP     GR1         恢复寄存器
        RET
DADD    ADD     GR0，B
        JMP     BACK
DSUB    SUB     GR0，B
        JMP     BACK
DAND    AND     GR0，B
        JMP     BACK
DOR     OR      GR0，B
        JMP     BACK
A       DS      1
B       DS      1
C       DS      1
D       DS      1
C2      DC      2
C4      DC      4
ENTRY   DC      DADD
        DC      DSUB
        DC      DAND        转移地址表
        DC      DOR
        END
```

[分析] 这是一个五分支结构的程序,按控制字 d 分情况转向某一程序段去计算.五个程序段的入口地址分别为:DADD、DSUB、DAND、DOR、DXOR.。如 DXOP 外，其它地址依次存放在首地址为 ENTRY 的散转表中,其存放顺序与控制字 d 在函数中取值的递增顺序相一致。

本程序首先判断控制字 d 是否在范围[2，5]之内。若在，则进行散转；否则转向标号为 DXOR 的程序段执行。程序中 GR1 寄存器存放控制字 d 且也作为 ENTRY 开始的散转地址表的指针，GR2 存放散转的有效地址。该程序的处理框图如图 7.6 所示。

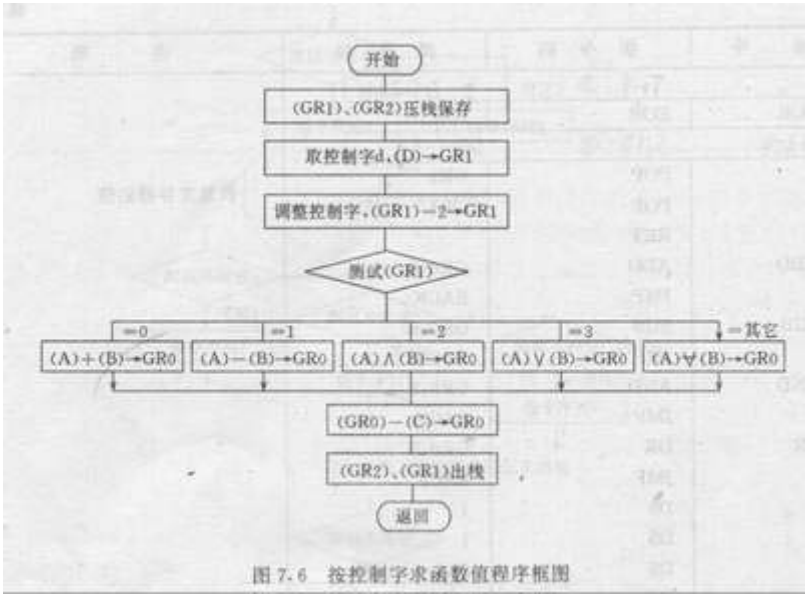


图 7.6 按控制字求函数值程序框图

[答案简析]

① 空应填指令：
SUB GR1，C2 或 LEA GR1，-2，GR1
因为①空下面一条指令为：JMI DXOR 且以后要根据 GR1 的值进行分情况转移。

② 空应填指令：
JMP 0，GR2
因为②空上面的一条指令为 LD GR2，ENTRY，GR1 将转移地址装入 GR2。因此该空应填指令 JMP 0，GR2。

③ 空应填指令：BACK SUB GR0，C

思考：如果将程序中 ENTRY 开始的散转地址表换成：
ENTRY JMP DADD
JMP DSUB
JMP DAND
JMP DOR

将程序中②空的前一条指令 LD GR2，ENTRY，GR1 换成 SLL GR1，1 则②空应填何指令？

在 CASL 程序中，转移指令转移的目标可以是在本程序内，也可以在另一个程序内。我们把转移目标在本程序内的转移称为程序内转移，把转移目标在另一个程序内的转移称为程序间转移。

[例 7.2]UPDATE 每隔一秒被调用一次，届时对存贮单元 HOUR（时），MINUTE（分）心及 SECOND（秒）的当前值予以更新处理。

SECOND 与 MINUTE 均为六十进制。HOUR、MINUTE 和 SECOND 的递加与复零均由公用子程序 ADDONE 进行。

[问题]填充下面程序中的 { }，完成程序，不必要的标号不要写。

[程序]

```
标号    指令码          操作数
UPDATE  START
        PUSH            0，GR1
        PUSH            0，GR2
        PUSH            0，GR3
        LEA             GR2,{①}
        LEA             GR3,{②}
        CALL            ADDONE
                { ③ }
                { ④ }
        CALL            ADDONE
                { ⑤ }
        LEA             GR2,{⑥}
        LEA             GR3,{⑦}
        CALL            ADDONE
UPTOK   POP             GR3
        POP             GR2
        POP             GR1
        RET
HOUR    DC              0
MINUTE  DC              0
SECOND  DC              0
ADDONE          { ⑧ }
        CPA             GR2,0,GR3
        JZE             ADDOK
        LD              GR1,0,GR3
        LEA             GR1,1,GR1
ADDOK   ST              GR1,0,GR3
        RET
        END
```

[分析]UPDATE 子程序，通过两个条件转移指令，实现多分支程序结构。调用 ADDONE 对 SECOND 加 1 后，判 SECOND 是否复零；若复零，则调 ADDONE 对 MINUTE 加 1，再判 MINUTE 是否复零；若复零，则调 ADDONE 对 HOUR 加 1。程序中各语句的具体分析说明如下：

```
标号    指令码          操作数          说明
UPDATESTART
```

```
        PUSH        0，GR0
        PUSH        0，GR2          GR1-GR3 的值压栈保存
        PUSH        0，GR3
        LEA         GR2，{ ① }
        LEA         GR3，{ ② }      对 SECOND 加 1
        CALL        ADDONE
                    { ③ }           若 SECOND 未复零，转 UPTOK
                    { ④ }           对 MINUTE 加 1
        CALL        ADDONE
                    { ⑤ }           对 MINUTE 未复零,转 UPTOK
        LEA         GR2，{ ⑥ }
        LEA         GR3，{ ⑦ }      对 HOUR 加 1
        CALL        ADDONE
PUTOK   POP         GR3
        POP         GR2             出栈，恢复 GR1-GR3 的值
        POP         GR1
        RET
HOUR    DC          0
MINUTEDC            0
SECONDDC            0
ADDONE              { ⑧ }           GR1 设置初值零
        CPA         GR2，0，GR3     若 SECOND 或 MINUTE
        JZE         ADDOK           或 HOUR 满足复零条件，则转 ADDOK
        LD          GR1，0，GR3     SECOND 或 MINUTE 或
HOUR 1LEA           GR1，1，GR1     加 1
ADDOK   ST          GR1，0，GR3     GR1 的值送 SECOND 或 MINUTE 或 HOUR 单元
        RET
        END
```

[答案简析]

① 空应填：59
因为 SECOND 为 60 进制，59 为 60 进制的最大数。

② 空应填：SECOND
因为下一条指令 CALL ADDONE，实现 SECOND 加 1，故 GR3 中应为 SECND 单元的地址。

③ 空应填指令：JNZ UPTOK
SECOND 未复零，则转移结束。

④ 空应填指令：LEA GR3，MINUTE
与下一指令 CALL ADDONE 结合实现 MINUTE 加 1。

⑤ 空应填指信令：JNZ UPTOK
MINUTE 未复零，则转移结束。

⑥ 空应填：23
因为 HOUR 为 24 进制，23 为 24 进制的最大数。

⑦ 空应填：HOUR
与下一指令 CALL ADDONE 结合，实现 HOUR 加 1。

⑧ 空应填指令：LEA GR1，0

7.2.2 循环

循环结构的特征是反复地执行一组指令，直到满足某种条件为止。循环结构一般都有 4 个组成部分：置初值、循环体、循环调整、循环控制。

（1） 置初值。在循环体开始执行前进行的初始化工作。它保证循环在正确的初始条件下工作。常做的初始化工作有：置操作数的首地址、置循环计数器的初值等。为了方便，我们把能控制循环结束的变量称为循环控制变量，在循环开始前要设置循环控制变量的初值。

（2） 循环体。循环体是循环结构的核心，则需要重复执行的若干操作组成。这此操作和运算是随求解问题的不同而各不相同的。每次循环，循环体一般都会在新的意义下产生一次中间结果。

（3） 循环调整。为了保证循环体每次都在新的意义下工作，循环体每执行一次都必须对操作对象、地址指针等进行一次调整，为下一次循环做准备。同时，循环计数器、循环控制变量等也要得到相应的修改或调整，以控制循环的继续或终止。

（4） 循环控制。对循环控制变量或循环计数器进行测试，以控制循环是继续还是终止。若达到预定的要求则循环结束；否则循环继续执行。

循环的控制形式有多种多样，常见的有两种形式：

① 计数控制，采用计数器控制循环。每循环一次，计数器递增或递增，当计数器达到某一值时，循环结束。这类循环的次数已知。

② 条件控制，这类循环次数未知，每执行一次循环后，都要测试是否满足循环结束的条件，若满足则结束循环；否则循环继续执行。

根据循环体、调整和控制的相对位置关系，循环的结构形式有六种，其中的四种如图 7.7 所示。

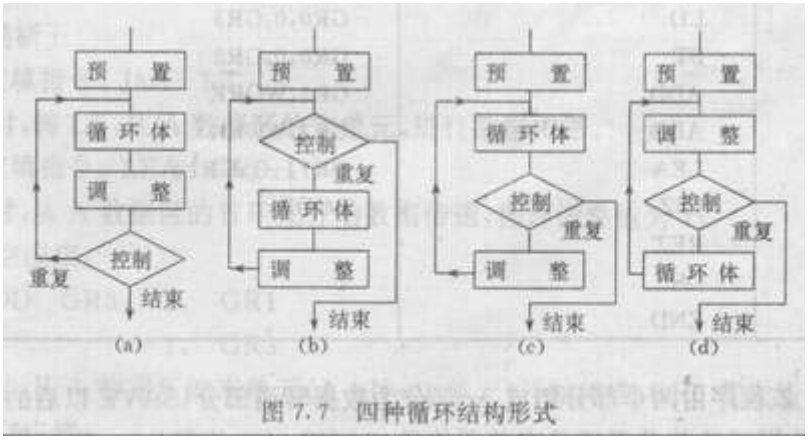


图 7.7 四种循环结构形式

有时问题比较复杂，用单循环结构还不够。于是就出现了循环体也是由一个循环结构组成的，即循环里面套循环。这种结构称为多重循环。循环常见的有二重或三重循环。在二重结构中，内循环的置初值部分是由外循环赋给的，它保证内循环每次都在一个正确的初始状态下运行。

[例 7.3]子程序 MOVE 是将地址为 A 开始的 N 个存储单元移动到地址为 B 开始的 N 个存储单元中,对于两个区域互相重叠的情况也能正确处理。主程序在 GR1 中给出存储子程序所需参数的起始地址。

参数的存放形式为：

(GR1)+0	A
+1	B
+2	N

[问题]填充下面程序中的（ ），完成程序，不必要的标号不要写。

[程序]

```
标号      指令码          操作数
MOVE      START
          LD              GR2，1，GR1
          LD              GR3，0，GR1
          CPL             GR3，1，GR1
          JZE             ENDMOV
                    { ① }
                    { ② }
          JMP             DAVE
LT                  { ③ }
          LEA             GR3，{④}
                    { ⑤ }
          LEA             GR2，{⑥}
                    { ⑦ }
```

```
SAVE    ST          GR0, WORK
        LD          GR1, 2, GR1
LOOP    LD          GR0, 0, GR3
        ST          GR0, 0, GR2
        ADD         GR2, WORK
        ADD         GR3, WORK
        LEA         GR1,-1,GR1
        JNZ         LOOP
ENDMOV  RET
WORK    DS          1
        END
```

[分析]该程序由两个部分组成。一部分为数据传送部分（SAVE 以后的部分）；另一部分确定从 A 数据区的首单元还是末单元传送（SAVE 以前的部分）。数据传送部分为一循环，SAVE 开始的两条指令设置循环调整值 1 或-1，设置循环次数 N。LOOP 开始的指令为循环每循环一次将 A 数据区一单元的值送至 B 数据区对应单元。程序中各指令的具体分析说明如下：

```
标号    指令码      操作数      说明
MOVE    START
        LD          GR2, 1, GR1B→GR2
        LD          GR3, 0, GR1A→GR3
        CPL         GR3, 1, GR1
        JZE         ENDMOV      A=B 转结束
                { ① }           A<B,转 LT
                { ② }           循环调整值 1→GR0
        JMP         DAVE        转 SAVE,进行数据传送
LT              { ③ }
        LEA         GR3, {④}    A+N-1→GR3
                { ⑤ }
        LEA         GR2, {⑥}    B+N-1→GR2
                { ⑦ }           循环调整值（-1）→GR0
SAVE    ST          GR0, WORK   循环调整值→WORK
        LD          GR1, 2, GR1 循环控制变量置初值
                                N→GR1
LOOP    LD          GR0, 0, GR3 数据传送
        ST          GR0, 0, GR2
        ADD         GR2, WORK   循环调整，调整 A,B 数
        ADD         GR3, WORK   据区指针
        LEA         GR1,-1,GR1
        JNZ         LOOP        循环次数<N,转 LOOP
ENDMOV  RET
WORK    DS          1
        END
```

[答案简析]

① 空应填指令：JMI LT
A＜B 时，转 LT，从 A 数据区的末单元，进行数据传送。

② 空应填指令：LEA GR0，1
A＞ B 时，从 A 数据区的首单元开始数据传送，循环调整值为 1。

③、④空应填：
ADD GR3，2，GR1
-1，GR3
A＜B 时，从 A 数据区的末单元（A+N-1 单元）开始传送，GR3 首先指向该单元。

⑤、⑥空应填：
ADD GR2，2，GR1
-1 GR2
A＜B 时，从 A 数据区的末单元开始传送，传至 B 数据区的末单元，该单元的地址为 B+N-1，GR2 首先指向该单元。

⑦空应填指令：LEA GR0，-1
A＜B 时，从 A 数据区的末单元开始传送，循环调整值为-1。

[例 7.4]冒泡排序程序。

一组有符号数 ai(i=1,2…,100),存放在 A1 开始的连续单元中，下面一程序将这组数在原来的内存区中由大到小重新排序。

[问题]填充下面程序中的（），完成程序，不必要的标号不要写。

[程序]

标号	指令码	操作数	说明
PRG	START		
	LEA	GR1,99	外循环设置初置
	ST	GR1,CN	
L1	LEA	GR1,0	
	LEA	GR2,1	内循环设置初置
	LEA	GR3,1	
L2	LD	GR0,A1,GR1	
	CPA	GR0,A1,GR2	
	JPZ	L3	
	LD	GR4,A1,GR2	内循环体
		{ ① }	
	ST	GR0,A1,GR2	
	LEA	GR3,0	

```
L3      LEA         GR1,1,GR1
        LEA         GR2,1,GR2     内循环调整
                { ② }
        JZE         L2            内循环控制
        JMI         L2
        CPA         GR3,C1
                { ③ }
        LD          GR4,CN        外循环调整
        LEA         GR4,-1,GR4
        JZE         L4            外循环控制
        ST          GR4,CN
        JMP         L1
L4      EXIT
A1      DS          100
CN      DS          1
C1      DC          1
        END
```

[分析]这是一个二重循环结构的程序。程序中采用冒泡排序的方法，首先从第 1 个单元开始，依次把相邻两个元素比较，即第 1 个单元内的数与第 2 个单内的数比较，第 2 个单内的数与第 3 个单元内的数比较，…第 99 个单元与第 100 个单元 的数相比较。每次把较大者放在前面单元，较小者放在后面单元。这样第一次遍历做了 n-1 次比较，最后最小的元素放在第 100 个单元。第二次遍历再从头开始，依次将相邻两个元素进行比较，把 n-1 个元素遍历一次需做 n-2 次比较，这样，第 99 个单元存放全部元素中次小元素。

如此反复，小的元素往下，大的元素向上犹如汽泡上浮，因此这种排队序的方法给它一个形象的名字，叫做冒泡算法。在程序中 GR1 指针，GR2 为内循环计数器兼数据区指针，GR3 为布尔寄存器。另外，还设外循环计数器 CN。程序处理框图如图 7.8 所示。

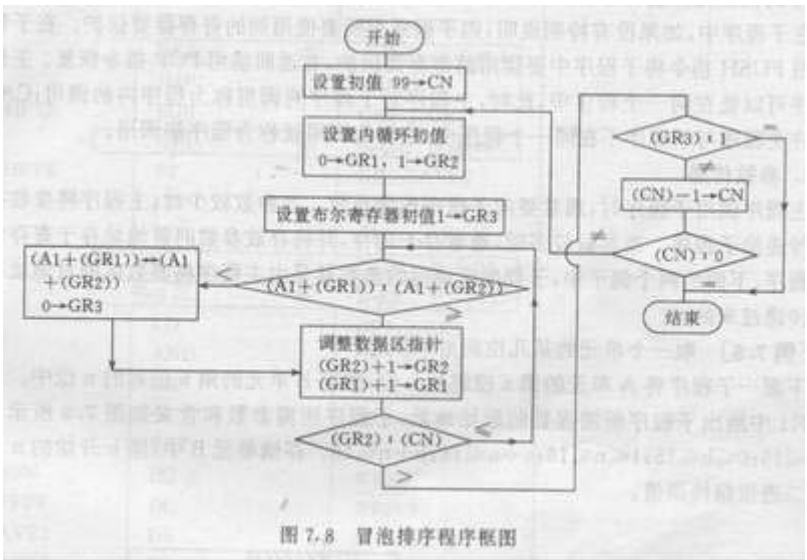


图 7.8 冒泡排序程序框图

[答案简析]

① 空应填指令：ST GR4，A1，GR1
前一条指令 LD GR4，A1，GR2 将 a_{i+1} 传到 GR4，要进行两数交换，则要把 a_{i+1} 存于原来 a_i 存放的单元。

② 空应填指令：CPA GR2，CN 或 CPL GR2，CN
前一条指令 LEA GR2，1，GR2 调整内循环指针，当内循环结束时 GR2 的值大于 CN 单元的值。因此该指令判断循环是否结束。

③ 空应填指令：JZE L4
上条指令 CPA GR3，C1 即（GR3）与 1 比较，若（GR3）=1，则表示前面的数已按从大到小的次序排好，则可转结束。

该程序是一个二重循环结构的程序。从图 7.8 可知，整个内循环（设置初值、循环体、循环调整和循环控制）是外循环的循环体，这就是所谓的循环里面套循环的结构。内循环和外循环都是计数控制的循环。除了排序程序是采用二重循环结构外，还有象矩阵操作等的程序也是采用二重循环结构。

7.2.3 主子程序

前面我们已接触到了一些子程序。所谓子程序是指完成一定功能的指令序列，在主程序的若干地方可以用 CALL 指令对它调用。子程序结束时通过 RET 指令返回主程序，子程序所处理的数据信息是由主程序传递过来的。主子程序之间的参数传递方式多种多样，经常采用的有，通过寄存器（GR0-GR3）传递参数值。通过寄存器（GR1-GR3）传递参数的地址，通过内存单元传递参数值和通过堆栈传递参数值。在子程充中一般不将 GR4 作为通用寄存器使用，因为 GR4 作为栈顶指针，如果其它通用寄存器不够用，要使用 GR4，则先将 GR4 中的栈顶指针值保存，然后使用 GR4，GR4 使用完后，立即恢复栈指针的值。

在子程序中，如果没有特别说明，则子程序中所要使用到的寄存器要保护。在子程序的开始用 PUSH 指令将子程序中要使用的寄存器保护，在返回前用 POP 指令恢复。主程序与子程序可以处在同一个程序中，此时，主程序对子程序的调用称为程序内的调用；CASL 中也允许主程序与子程序不在同一个程序中，这样的调用就称为程序间调用。

1. 参数传递

主程序调用子程序时，通常要向子程序传递参数。当参数较少时，主程序将参数存于寄存器传给子程序。当参数较多时，通常存于内存，并将存放参数的首地址存于寄存器传递给子程序。下面的两个例子中，子程序所使用的参数就是由主程序将参数区的首地址存于寄存器传递过来的。

[例 7.5]取一个单元的某几位到另一单元。

下面一子程序将 A 单元的第 a 位起的 n 位，搬至 B 单元的第 b 位起的 n 位中。主程序在 GR1 中给出子程序所需参数的起始地址，子程序所需参数和含义如图 7.9 所示。其中 0≤a≤15;0≤b≤15;1≤n≤16;a+n≤16;b+n≤16。存储单元 B 中，除 b 开始的 n 位外的其它二进制位保持原值。

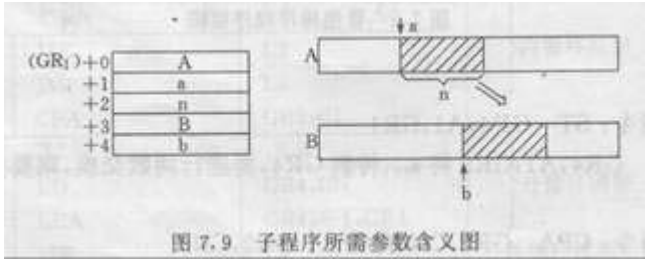


图 7.9 子程序所需参数含义图

[问题]填充程序中的（ ），完成程序，不必要的标号不要写。

[程序]

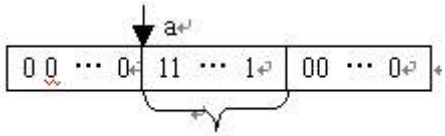
```
标号    指令码      操作数      说明
MOVBIT  START
        LD          GR2,2,GR1   在中 SAV1 形成
                { ① }
                { ② }
        ST          GR0,SAVE1
        LD          GR2,1,GR1
                { ③ }
        LD          GR2,0,GR1
        AND         GR0,0,GR2
        LD          GR2,4,GR1
        CPA         GR2,1,GR1
                { ④ }
        LD          GR2,1,GR1
        SUB         GR2,4,GR1
                { ⑤ }
        JMP         SHFTE
        SUB         GR2,1,GR1
                { ⑥ }
        ST          GR0,SAVE2
        LD          GR2,SAVE1
        LD          GR3,4,GR1
                { ⑦ }
                { ⑧ }
        LD          GR3,3,GR1
        AND         GR2,0,GR3
        OR          GR2,SAVE2
        ST          GR2,0,GR3
```

11 … 1	0 0 … 0

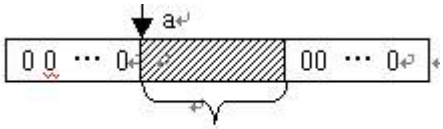
n 位

```
        RET
        DC      #8000
        DC      #FFFF
        DS      1
        DS      1
        END
```

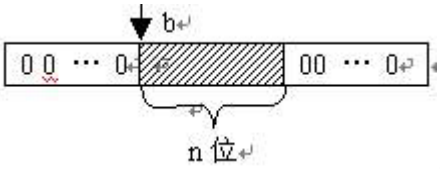
[分析]主程序通过寄存器 GR1 存放参数的首地址，将参数传送给子程序。程序的结构是一个分支结构，移位指令是这个程序的关键指令。程序中为了析取 A 单元



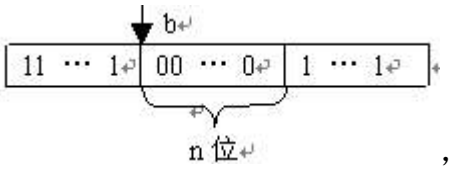
第 a 位开始的 n 位，构造了逻辑字 ，用此逻辑字



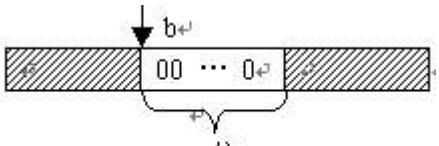
和 A 单元的内容进行逻辑乘即得 A ，然后对此左



移或右移得 。程序中还构造了一个逻辑字



，与 B 单元的值逻辑乘，将 B 单元中从第 b 位开



始的 n 位屏掉，得到 B ，此值和 A 进行逻辑加即得所求。程序

框图如图 7.10 年示。

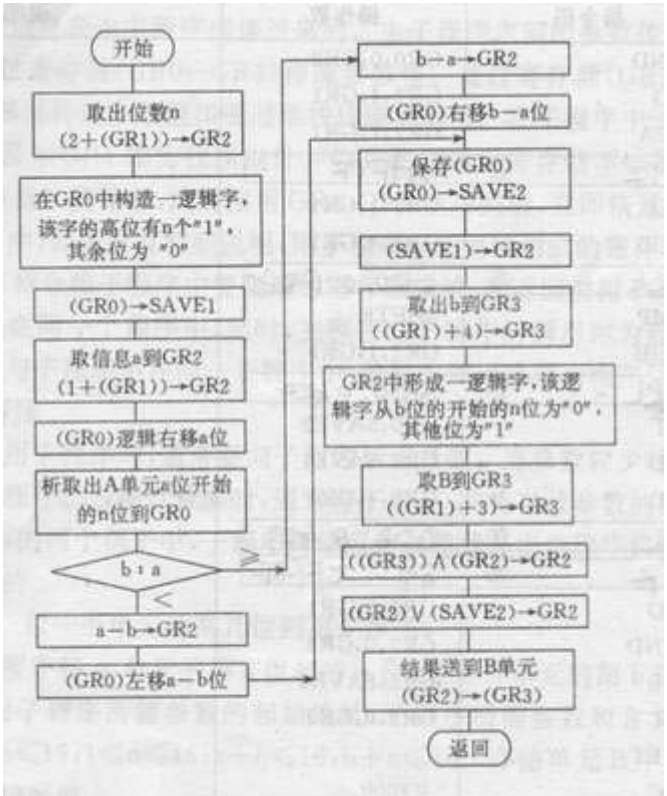
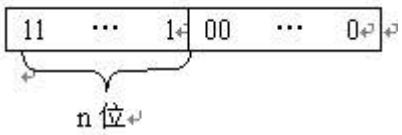


图 7.10 取一个单元的某几位到另一个单元的程序框图

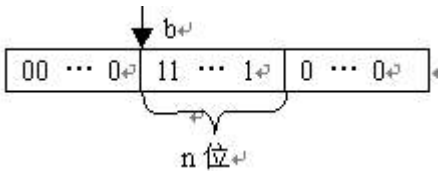
[答案简析]

① 空应填指令：LD GR0，C8000 或 LEA GR0，-32768 或 LEA GR0，32768

② 空应填指令：SRA GR0，-1，GR2
这两条指令为了在 GR0 中构造逻辑字：



③ 应填指令：SRL GR0，0，GR2 构造逻辑字：

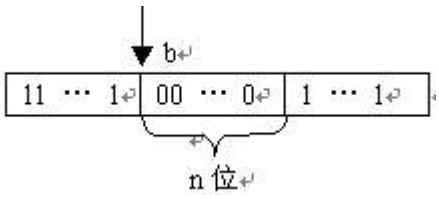


④ 空应填指令：JPZ SHFTR b≥a 时转 SHFTR 将（GR0）右移 b-a 位。

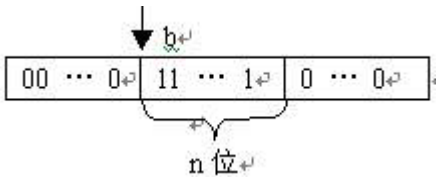
⑤ 空应填指令：SLL GR0，0，GR2 使（GR0）左移 a-b 位。

⑥ 空应填指令：SRL GR0，0，GR2 使（GR0）右移 b-a 位。

⑦ 空应填指令：SRL GR2，0，GR3 形成逻辑字：



⑧ 空应填指令：EOR GR2，CFFFF 形成逻辑字：



在该子程序中，GR1 存放的是参数区的首地址。子程序要使用参数时，将 GR1 作为变址器使用访问参数。如程序中的指令：LD GR2 ，2，GR1、LD GR2，1，GR1、LD GR2，4，GR1 和 CPA GR2，1，GR1 等都是访问参数的指令。

[例 7.6]无符号乘法。

有两个无符号数 N1、N2，下面一子程序实现 N1×N2 返回。调用子程序时 GR1 存放参数 N1、N2 和相乘结果的存放区域的首地址，如图 7.11 所示。设相乘过程不会产生溢出。



图 7.11 无符号乘法参数图

[问题]填充下面程序中的（ ），完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数	说 明
MULT	START		
	ST	GR0,SAVE	保护寄存器

	PUSH	0,GR2	
	PUSH	0,GR3	
	LEA	GR2,0	
	LEA	GR3,16	
	{ ① }		取出乘数
LOOP	SLL	GR2,1	
	SLL	GR0,0	
	JPZ	L1	
	{ ② }		
L1	SLL	GR0,1	
	LEA	GR3,-1,GR3	
	JNZ	LOOP	
	{ ③ }		存放乘积
	LD	GR0,SAVE	
	POP	GR3	恢复寄存器的值
	POP	GR2	
	RET		
SAVE	DS	1	
	END		

[分析]像手算乘法一样，由于两个数是二进制数，所以唯一的问题是乘 0 还是乘 1；显然，乘 0 结果为 0，而乘 1 则结果与开始的数（被乘数）相同。因此在二进制乘法中每一步都可以简化成下列运算：如果乘数中现有位为 1，则将被乘数加到部分积上;为 0 则不加,剩下的问题是部分积和被乘数每次相加要进行移位，保证每次相加都正确对准。乘法过程可分成以下几步：

（1） 积清 0，设置计数器初值 16。

（2） 积左移 1 位。

（3） 乘数最高位是否为 1。

（4） 若为 1，则被乘数加积。

（5） 乘数左移 1 位，计数器减 1，若不为 0 转（2）。

程序中 GR0 存乘数，GR2 存积，GR3 为计数器，程序处理框图如图 7.12 所示。

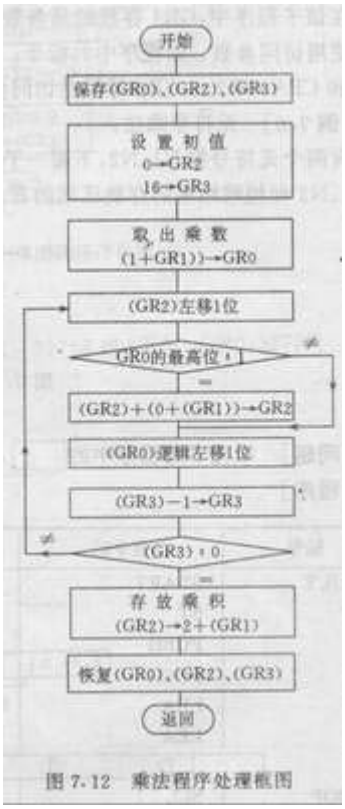


图 7.12 乘法程序处理框图

[答案简析]

① 空应填指令：LD GR0，1，GR1 取出乘数到 GR0。从后面的指令得知 GR0 中存放的是乘数。

② 空应填指令：ADD GR2，0，GR1 当乘数的最高位为 1 时，部分积加被乘数。

③ 空应填指令：SL GR2，2，GR1 在返回主程序前将乘积存于参数区。

2. 子程序的返回

子程序通过 RET 指令返回主程序。RET 指令的功能是将栈顶的返回地址置入 PC。CASL 中，GR4 兼作通用寄存器、变址器和栈顶指针，因此在子程序中可以通过 GR4 修改返回地址，这样子程序返回时，就不一定返回到主程序的调用点下面的一条指令。在 CASL 程序设计中常常利用这一技巧。

[例 7.7]判断字符串是否为某程序设计语言的变量名。

下面的程序是在输入装置上读入一个字符串，判断它是否为合法的某程序设计语言的变量名，并区分该变量名的类型（实型或整型）。判断或区分的规则如下：

① 变量名的第一个字符必须是英文大写字母，从第二个字符起可以是英文大写字母或数字。

② 变量名只能用 6 个以内的合法字符表示（包含 6 个合法字符）。

③ 变量名的第一个字符为英文大写字母 I，J，K，L，M，N 时，该变量名的类型为整型，其它英文大写字母开始的变量名的类型为实型。

④ 输入不符合上述规则的字符串，程序输出*ERR。

输入字符串和输出结果的关系举例如下：

输入字符串	输出字符串
OR	REAL
INTEN	INTE
4BAS	*ERR
FORTRAN	*ERR

[问题]填充下面程序中的（ ），完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数
	START	BEG
BEG	IN	INBUF,INNUM
	OUT	INBUF,INNUM
	LEA	GR1,6
	CPA	GR1,INNUM
	JMI	ERR
	LEA	GR1,0
	CPA	GR1,INNUM
	JPZ	ENDD
AGN	LD	GR3,INBUF,GR1
	CALL	AZ09IN
	DC	0
	JMP	NEXT
	LEA	GR1,0,GR1
	JZE	ERR
	CALL	AZ09IN
	DC	2
	JMP	NEXT
ERR	OUT	EMSG,N4
	JMP	BEG
NEXT	LEA	GR1,1,GR1
	CPA	GR1,INNUM

```
        JMI     AGN
        { ① }
        { ② }
        { ③ }
        { ④ }
        OUT     RMSG,N4
        JMP     BEG
INT     OUT     IMSG,N4
        JMP     BEG
ENDD    EXIT
AZ09IN  LD      GR2,0,GR4
        LD      GR2,0,GR2
        CPA     GR3,CHAR,GR2
        JMI     NO
        LEA     GR2,1,GR2
        CPA     GR3,CHAR,GR2
        JMI     YES
        JNZ     NO
YES     { ⑤ }
        { ⑥ }
        ST      GR2,0,GR4
        RET
NO      { ⑦ }
        { ⑧ }
        ST      GR2,0,GR4
        RET
N4      DC      4
EMSG    DC      '*ERR'
RMSG    DC      'REAL'
IMSG    DC      'INTE'
INNUM   DS      1
INBUF   DS      80
CHAR    DC      'AZ09IN'
        END
```

[分析]

根据变量名的构成规则判断输入的字符串是否是合法的变量名，对合法的变量名应区分是整型还是实型。先判断字符串的长度是否超过 6，然后判断打头的是否

是英文大写字母，再判断后面跟的是否是英文大写字母或数字，最后再确定是整型还是实型变量。对程序中各语句的具体分析说明如下：

标 号	指 令 码	操 作 数	说 明
	START	BEG	输入字符串
BEG	IN	INBUF,INNUM	输出字符串
	OUT	INBUF,INNUM	字符串长度大于 6 转 ERR
	LEA	GR1,6	
	CPA	GR1,INNUM	
	JMI	ERR	字符串长度等于零转 ENDD
	LEA	GR1,0	
	CPA	GR1,INNUM	
	JPZ	ENDD	
AGN	LD	GR3,INBUF,GR1	取一字符到 GR3
	CALL	AZ09IN	调 AZ09 子程序判是否为大写英文字母，是转 NEXT
	DC	0	
	JMP	NEXT	
	LEA	GR1,0,GR1	打头的不是大写英文字母转 ERR
	JZE	ERR	
	CALL	AZ09IN	调 AZ09 子程序判是否为数字，是转 NEXT
	DC	2	
	JMP	NEXT	
ERR	OUT	EMSG,N4	输出出错信息，转 BEG
	JMP	BEG	
NEXT	LEA	GR1,1,GR1	调整 INBUF 数据区指针
	CPA	GR1,INNUM	判断指针是否出界，未出界转 AGN
	JMI	AGN	
		{ ① }	判断输入字符串的打头字符是否是 I-N 的大写字母，是转 INT
		{ ② }	
		{ ③ }	
		{ ④ }	
	OUT	RMSG,N4	输出’REAL’信息，转 BEG
	JMP	BEG	
INT	OUT	IMSG,N4	输出’INTE’信息，转 BEG
	JMP	BEG	
ENDD	EXIT		
AZ09IN	LD	GR2,0,GR4	取调用指令下面的常数到 GR2
	LD	GR2,0,GR2	
	CPA	GR3,CHAR,GR2	
	JMI	NO	判断 GR3 中的字符是否在对

	LEA	GR2,1,GR2	应的范围内，是转 YES,不是转 NO
	CPA	GR3,CHAR,GR2	
	JMI	YES	
	JNZ	NO	
YES		{ ⑤ }	返回地址增 1，指向调用 AZ09IN 子程序下面的常数伪指令下面一条指令
		{ ⑥ }	
	ST	GR2,0,GR4	
	RET		
NO		{ ⑦ }	返回地址增 3，GR3 中的字符不在指定的范围内
		{ ⑧ }	
	ST	GR2,0,GR4	
	RET		
N4	DC	4	
EMSG	DC	'*ERR'	
RMSG	DC	'REAL'	
IMSG	DC	'INTE'	
INNUM	DS	1	
INBUF	DS	80	
CHAR	DC	'AZ09IN'	
	END		

[参考答案]

① 空可填指令：LD GR3，INBUF
取出输入的第一个字符，判断是否是 I-N 之间的字符。

② 空可填指令：CALL AZ09IN
判断 GR3 中的字符是否为 I-N 之间的字符。

③ 空可填指令：DC4 或 DC #0004

④ 空可填指令：JMP INT
转 INT 输出'INTE'信息。

⑤ 空可填指令：LD GR2，0，GR4
取出返回地址。

⑥ 空可填指令：LEA GR2，1，GR2
调整返回地址。

⑦ 空可填指令：LD GR2，1，GR4
取出返回地址。

⑧ 空可填指令：LEA GR2，3，GR2
调整返回地址。

从上面的分析说明可以看出，⑤⑥空与⑦⑧空均为调整返回地址的指令。因为主程序通过 CALL 指令对子程序调用时，将返回地址压栈保存。在子程序中可通过栈指针 GR4 取出返回地址，对其修改，然后再存入栈中，子程序通过 RET 指令返回时，将返回到修改后的新地址处。

3. 子程序的调用

在 CASL 中，主程序通过 CALL 指令对子程序调用。通常子程序都只有一个入口点，如上例的 AZ09IN。但 CALL 指令允许变址调用，提供了子程序可以有多个入口点之便。这样主程序就可以根据不同的情况，进入同一子程序的不同入口点。

[例 7.8]求函数 f（x）的值存于 GR1。

f(x)=

a-1+c x=0 时
a+1+c x=1 时
a+b+c x=2 时
a-b+c x=其它时

其中 a、b、c、x 均为非负整数，分别存于 A、B、C、X、单元。

[程序]

标 号	指 令 码	操 作 数	说 明
M	START		
	LD	GR1,A	
	LD	GR2,X	
	LEA	GR3,-3,GR2	
	JMI	L1	
	LEA	GR2,3	
L1	LD	GR2,TABLE,GR2	取子程序的入口点
	CALL	0,GR2	
	EXIT		
TABLE	DC	SUB1	
	DC	SUB2	
	DC	SUB3	
	DC	SUB4	
SUB1	LEA	GR1,-1,GR1	入口点 1

	JMP	RT	
SUB2	LEA	GR1,1,GR1	入口点 2
	JMP	RT	
SUB3	ADD	GR1,B	入口点 3
	JMP	RT	
SUB4	SUB	GR1,B	入口点 4
RT	ADD	GR1,C	
	RET		
A	DS	1	
B	DS	1	
C	DS	1	
X	DS	1	
	END		

[分析]这是一个主子结构的程序，主程序利用 CALL 指令允许变址调用的特点，按情况进入子程序的不同入口点，子程序的四个入口点 SUB1、SUB2、SUB3、SUB4 依次构造一入口地址表 TABLE，主程序根据控制字（X 单元的内容），从 TABLE 表中取得相应的入口地址，然后调用了程序。

如果将该程序中的入口地址表：

```
TABLE   DC      SUB1
        DC      SUB2
        DC      SUB3
        DC      SUB4
```

换成：

```
TABLE       JMP     SUB1
            JMP     SUB2
            JMP     SUB3
            JMP     SUB4
```

将程序中 CALL 0，GR2 上面的一条指令：LD GR2，TABLE ，GR2 换成 SLL GR2，1 将 CALL 0，GR2 换成 CALL TABLE，GR2 该程序也正确。

与 7.2.1 中介绍的分支情况的多分支结构一样，CASL 也提供了按情况进入了程序的不同入口点的功能。所不同的是按情况进入了程序的不同入口点，从子程序返回时，都返回到调用点的下面一条指令。对于这一类程序关键是按控制字递增的顺序构造成入口地址表。

4. 通用寄存器保护

通常情况下，子程序都要保护它所使用到的寄存器，除非有特别说明，子程序可以不保护地使用寄存器。通常保护寄存器的方法是，在子程序开始时用 PUSH 指令保护 GR1-GR3，用 ST 指令保护 GR4、GR0，在子程序结束时用 POP 指令恢复 GR1-GR3，用 LD 指令恢复 GR0。

如[例 7.6]中，MULT 子程序开始部分用 ST 指令保存 GR0 的值，用 PUSH 指令保存 GR2、GR3 的值，在子程序结束时，用 LD 指令恢复 GR0 的值，用 POP 指令恢复 GR2、GR3 的值。

7.2.4 程序举例

前面介绍了 CASL 程序的基本结构。CASL 程序都是由这些基本结构成的。在 CASL 程序中有一类对字符串进行处理的程序，称为字符串处理程序。对于这一类程序首先要了解字符串在内存的存储形式。采用输入宏指令 IN 和字符串常数定义伪指令 DC 定义这字符串存于一段连续的内存区，且一个字符占一个单元，字符的 ASCII 码存于低八位，高八位全为 0。要掌握字符串处理程序，有必要了解常用字符的 ASCII 码，并熟悉 ASCII 码表。如，数字字符 0-9 的 ASCH 码为十六进制的 30-39。大写字母 A-Z 的 ASCII 码为十六进制的 41-5A。小写字母'a'-'z'的 ASCH 码为十六进制的 61-7A。在 ASCII 码表中排在字符'9'后面的字符是'：'；排在字符'Z 后面的字符是'['；排在小写字符'z'后面的字符是'{'；空格字符的 ASCII 码为十六进制的 20。

字符串处理程序的结构一般都含有循环和分支结构。由于字符在内存是以字符数据（ASCH 码）形式存放的，字符串处理程序类似于对数组处理的程序，程序中都含有存放字符串的数据区，这一类程序总是扫描该数据区。

[例 7.9]删去指定的字符。

BUF 开始的区域存放一字符串，每个字符存于一个单元的低八位，高八位全 0，字符串长度存于 LEG 单元。下面一子程序从 BUF 开始的字符串中删去 GR0 中的字符。

[问题]填充下面程序中的（ ），完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数	说 明
QUEEZE	START		
	PUSH	0,GR1	保存寄存器的值
	PUSH	0,GR2	
	PUSH	0,GR3	
	LEA	GR2,0	设置初值
	LEA	GR1,-1	
LOOP	LEA	GR1,1,GR1	循环调整

	CPA	GR1, LEG	循环控制
	{ ① }		
	CPA	GR0, BUF, GR1	循环体
	JZE	LOOP	
	LD	GR3, BUF, GR1	
	ST	GR3, BUF, GR2	
	{ ② }		
	JMP	LOOP	
RT	ST	GR2, LEG	
	{ ③ }		恢复寄存器
	{ ④ }		
	{ ⑤ }		
	RET		
BUF	DS	80	
LEG	DS	1	
	END		

[分析]程序中采用了两个指针 i 和 j,i 指针扫描整个字符串。该程序的结构是一循环结构，每次循环从字符串中取出一字符，判断是否是要删除的字符。若是，则 i 指针向下移一个单元。循环结束后还要把删除掉指定的字符后的新的字符串的长度存入 LEG 单元。程序处理品框图如图 7.13 所示。程序中 GR1 作为 i 指针，GR2 作为 j 指针。

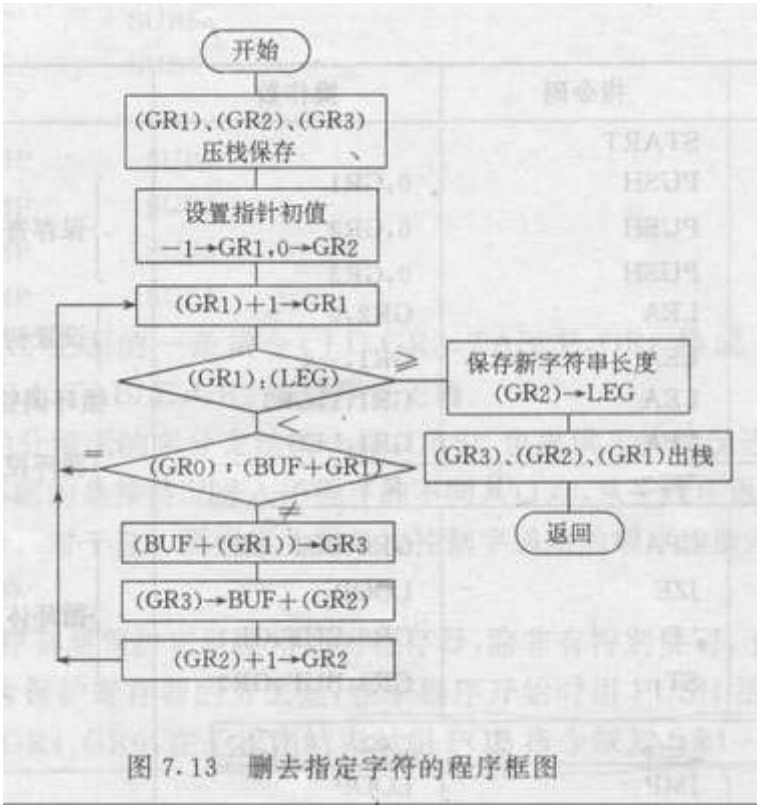


图 7.13 删去指定字符的程序框图

[答案简析]

① 空应填指令：JPZ RT 或 JZE RT
因为上一条指 CPA GR1，LEG 判断是否扫描完整个字符串，若扫描完，则转返回处理。当扫描完整个字符串时，（GR1）≥（LEG）

② 空应填指令：LEA GR2，1，GR2
调整 j 指针。当字符传送后要使 j 指针指向下一单元。

③、④、⑤空应填指令：
POP GR3
POP GR2
POP GR1
和程序开始部分的压栈指令相对应。

[例 7.10] 去掉数字中开头的无效零。

下面一程序从终端输入设备上输入八位数字，对这八位数字去掉开头的无效零，并输出处理结果。其处理过程就是把头一个非零数字前面的零全部去掉，从头一个非零数字开始显示。

例如：输入 00020040
输出 20040

输入数字不是八位时，显示出错信息，并要求重新输入。8 位都是 0 时，只显示最后一个 0。只输入回车符时，程序结束。不输入数字以外的字符。

[问题]填充下面程序中的（ ），完成程序，不必要的标号不要写。

标 号	指 令 码	操 作 数	说 明
PRG	START	BEGIN	
BEGIN	IN	INBUF,LENG	输入数据
	LD	GR3,LENG	
	CPA	GR3,C0	输入的是回车符号吗？
	JZE	LAST	
	CPA	GR3,C8	输入位数=8 吗？
	JZE	GOOD	
	OUT	ERMES,C11	
	JMP	BEGIN	
	{ ① }		设置初值
	LD	GR1,C0	
LOOP	CPA	GR1,C7	
	JPZ	OUTPUT	

	CPL	GR0,INBUF,GR1	
	JNZ	OUTPUT	
	LEA	GR1,1,GR1	
	JMP	LOOP	
		{ ② }	
LOOP2	CPA	GR1,C8	
	JPZ	PRINT	
	LD	GR0,INBUF,GR1	
		{ ③ }	
	LEA	GR1,1,GR1	
	LEA	GR2,1,GR2	
	JMP	LOOP2	
PRINT	ST	GR2,OLNG	
	OUT	OUTBUF,OLNG	显示结果
	JMP	BEGIN	
LAST	EXIT		
INBUF	DS	80	
LENG	DS	1	
OUTBUF	DS	8	
OLNG	DS	1	
ZERO	DC	'0'	
C0	DC	0	
C7	DC	7	
C8	DC	8	
C11	DC	11	
ERMES	DC	'INPUT ERROR'	
	END		

[程序]

[分析]输入数字后该程序首先检查是否回车符号，再检查输入是否为八个字符，然后对输入的字符逐个检查，看是否为 0，若是，则检查下个字符直到一个不为 0 的字符或到最后一个字符为止。最后把输入数据区剩余的字符传到输出数据区进行输出。程序中 GR1 作为循环计数器，GR2 统计第一个非零字符后的字符个数。程序处理框图如图 7.14 所示。

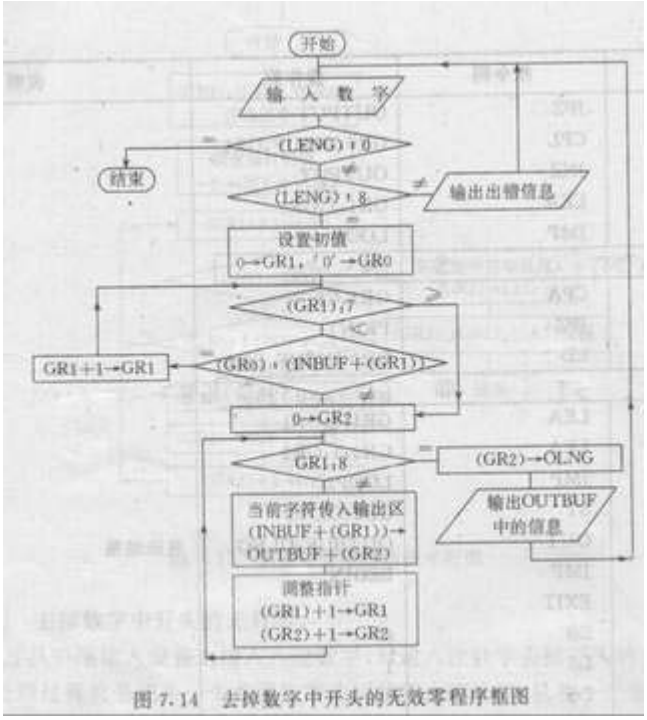


图 7.14 去掉数字中开头的无效零程序框图

[答案简析]

① 空应填指令:GOOD LD GR0,ZERO 或 GOOD LEA GR0,48
在进行循环前将字符'0'送入 GR0。

② 空应填指令：OUTPUT LEA GR2，0 或 OUTPUT GR2，C0
GR2 用于统计第一个非零字符后的字符个数。在进入 LOOP2 循环前，要给 GR2 赋初值。

③应填指令：ST GR0，OUTBUF，GR2
因上一条指令为 LD GR0，INBUF，GR1 从输入数据区取出第一个非零数字字符后的字符，该空的指令将它传至输出数据区。

CASL 中还有一类程序称之为输入输出程序，输入与输出是由 IN 和 OUT 宏指令实现的，因此不必考虑计算机和外设的接口，也不必考虑对外设的寻址。CASL 中的输入程序主要考虑如何将 IN 输入的数据转换成机内的数值，输出程序则正好相反。

7.3 CASL 与数据结构

上节介绍了一般的程序设计技术，本节结合汇编程序经常应用的数据结构，进一步介绍程序设计方法。从本质上看，程序的运行就是对数据的处理，被处理的数

据在机内的存储和组织形式就称之为数据结构。汇编中使用的数据结构有：数组，队列，栈、链表和二叉树等。数组在 CASL 试题中经常使用。

7.3.1 数组

数组一般分为一维数与二维数组等。一维数组的各元素是存于一段连续的存储区中的。如数组 A（10）是由 10 个元素组成的一维数组，这 10 个元素为 a1,a2,…，a10，数组 A（10）存于以 A 单元开始的区域。如图 7.15 所示。

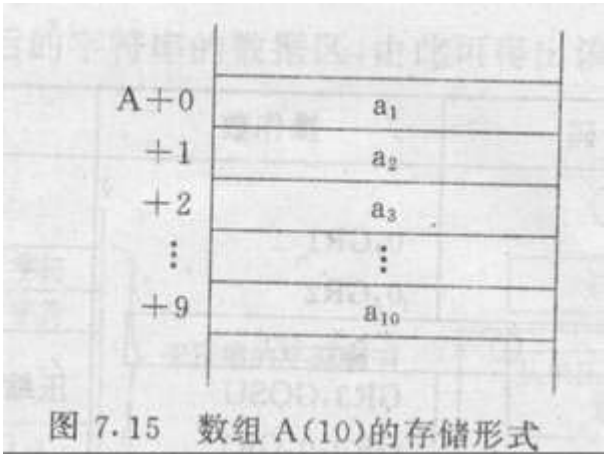


图 7.15 数组 A(10)的存储形式

指令 LEA GR1，0
LD GR0 A，GR1

将数组元素 a1 取入 GR0，通过 LEA 指令调整 GR1 的值，可访问数组 A 中各元素。CASL 试题中普遍使用一维数组，如用于存放输入数据的数据区，主程序调用子程序时用于参数传送的数据区。例如，一主程序调用子程序时，将子程序所需的 5 个参数 N1，N2…，N5 存于 5 个连续单元组成的数据区，GR1 中存放数据区的首地址，如图 7.16 所示。

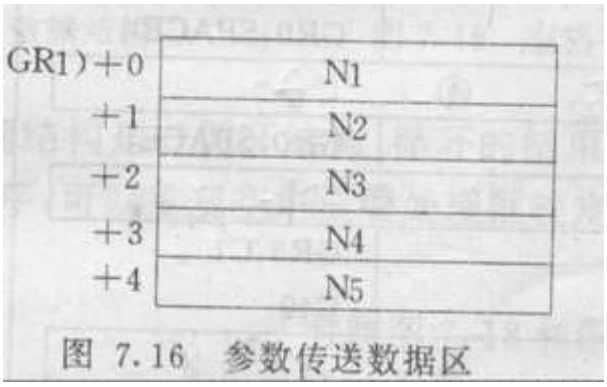


图 7.16 参数传送数据区

在子程序中，GR1 作为变址寄存器，可访问参数区中任一参数。如，指令 LD GR0，0 GR1 将参数 N1 传入 GR0。在下面的例子中，存放字符串的数据区的首单元的地址就存于 GR1。

[例 7.11]字符串压缩。

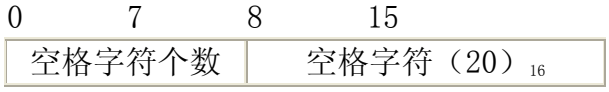
一子程序的功能是将字符串中两个以上的连续空格字符，压缩成一个字符。

（1） 字符串放在由 GR1 所指出地址为开始的连续内存区域内；各单元放一个字符；各个单元的 0-7 位全为 0，8-15 位为字符。

（2） 字符串的长度在 100 以下，字符串最后放（FFFF）16。

（3） 压缩后的字符串放在由 GR2 所指出地址为开始的连续内存区域内。GR3 放压缩后字符串的字符数。然后返回主程序。

（4） 压缩的空格字符如下所示。但空格字符是一个字符的情况下，空格字符数为 0。



[问题]填充下面程序中的（ ），完成程序，不必要的标号不要写。

```
标号    指令码      操作数    说明
COMP    START
        PUSH        0,GR1
        PUSH        0,GR2
                {  ①  }
        ST          GR3,GOSU 压缩后的字符数初始化
C10     LD          GR0,0,GR1
        CPA         GR0,OWARI字符串是否结束？
        JZE         C50
        CPL         GR0,SPACE是不是空格字符？
                {  ②  }
        LEA         GR3,0
C20             {  ③  }
        LEA         GR1,1,GR1
        LD          GR0,0,GR1
        CPL         GR0,SPACE是否空格字符？
                {  ④  }
        LD          GR0,SPACE
C30             {  ⑤  }         空格个数是 1 吗？
        CPA         GR3,C1
        JZE         C40
                {  ⑥  }
        OR          GR3,SPACE
        LEA         GR0,0,GR3
C40     LD          GR3,GOSU
```

```
        LEA     GR3,1,GR3压缩后的字符数加 1
        ST      GR3,GOSU
        ST      GR0,0,GR2
        LEA     GR2,1,GR2压缩后的字符串指针加
                        1
        LEA     GR1,1,GR1字符串指针加 1
                { ⑦ }
C50     LD      GR3,GOSU
        POP     GR2
        POP     GR1
        RET
SPACE   DC      #0020
C1      DC      1
OWAR1   DC      #FFFF
GOSU    DS      1
        END
```

[分析]阅读程序说明可知，该程序处理过程中，所使用的数据结构如图 7.17 所示。

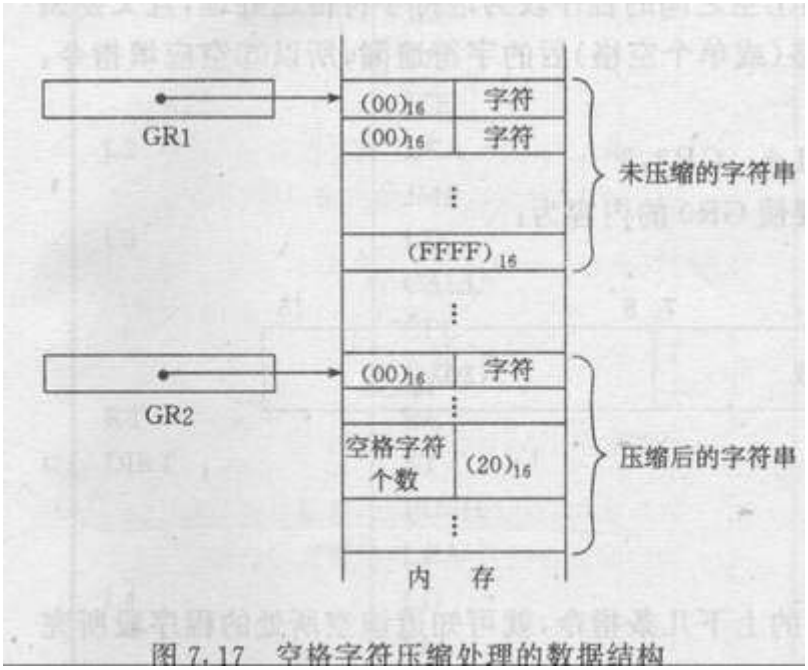


图 7.17 空格字符压缩处理的数据结构

从程序说明可知，空格字符压缩程序只对连续的空格字符时行压缩处理，对非空格字符不作任何处理传至存放压缩后的字符串的数据区，由此可得出该程序的处理框图如图 7.18 所示。

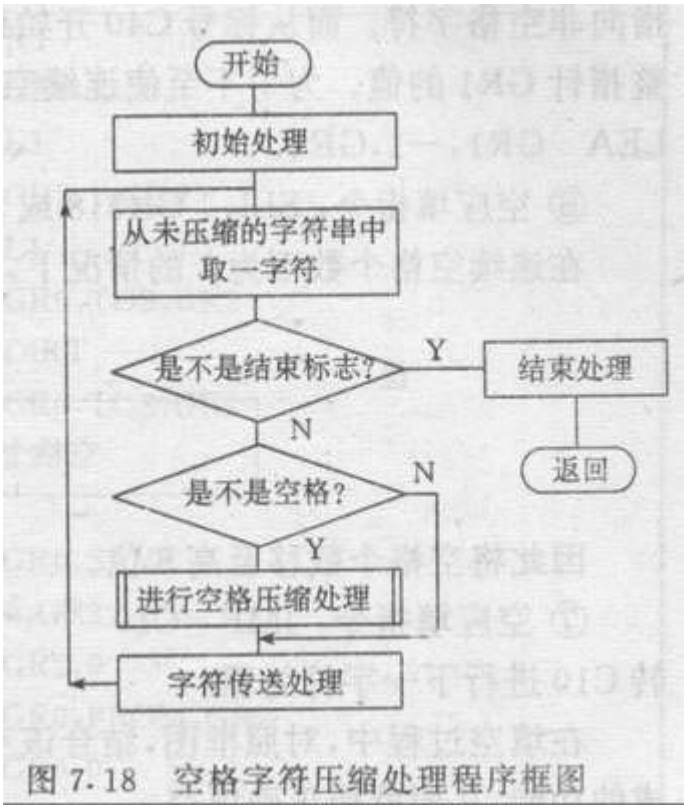


图 7.18 空格字符压缩处理程序框图

从图 7.18 可以看出整个处理结构是一循环结构,循环的结束条件为取出一结束标志(FFFF)16。对照程序框图阅读程序,可以发现空格压缩处理较复杂,细化成如图 7.19 所示。



图 7.19 空格压缩处理细化框图

整理人：刘海云
整理时间：2003.05

对照图 7.18 和图 7.19 并阅读程序,得出以下的答案。

[答案简析]

① 空应填指令：LEA GR3，0
因为下一条指令为 ST GR3，GOSU;
其中 GOSU 单元记录压缩后的字符个数。故初始时要给 GOSU 单元设置初值 0。

② 空应填指令：JNZ C40
因为上一条指令为：CPL GR0，SPACE 判断所取的字符是否为空格，如不是则进行字符传送处理。程序中从标号 C40 开始的部分为字符传送处理部分。

③ 空应填指令：LEA GR3，1，GR3
GR3 在空格压缩处理部分作为空格计数器，上一条指令为 LEA GR3，0；设置 GR3 的初值，因此该条指令调整 GR3 的值。

④ 空应填指令：JZE C20
因上一条指令 CPL GR0，SPACE 判断取出的字符是否为空格。从标号 C20 到空④的程序段为统计连续空格的个数。因此 GR0 中若为空格字符，则空格计数器就要增 1。

⑤ 空应填指令：LEA GR1，-1，GR1
调整未压缩的字符串的指针 GR1 的值。因为经过统计连续空格个数的处理后,GR1 已指向非空格字符。而从标号 C40 开始到⑦空之间的程序段为进行字符传送处理，且又要调整指针 GR1 的值。为了不至使连续空格（或单个空格）后的字符遗漏，所以⑤空应填指令：LEA GR1，-1，GR1。

⑥ 空应填指令：SLL GR3，8，或 SLA GR3，8
在连续空格个数不为 1 的情况下，要使 GR0 的内容为：

0 7	8 15
空格个数	$(20)_{16}$

因此将空格个数移至 8 位。

⑦ 空应填指信令:JMP C10
转 C10 进行下一字符处理..

在填空过程中,对照框图,结合该空的上下几条指令,就可知道该空所处的程序段所完成的功能,从而就能正确填空.

有时,根据留空前后的指令,也可以猜出应填什么指令.一般情况下,比较指令的下一条指令为条件转移指令.如本例中的②空和④空的前一条指令都为比较指令,可以猜出②空和④空应填条件转移指令.

① 空的下一条指令对 GR3 进行引用,①空前面也示出现过给 GR3 赋值的指令.因此①空所填指令要给 GR3 赋值.

[例 7.12] 查找指定两个城市间的航线。

有 20 个城市通航(20 个城市用 1～20 的编号表示).如果甲城到乙城有直接航班就用一对数表示.如,:"3.7"表示 3 号城到 7 号城有直接航班.20 个城市的所有直接航班信息记录在 FROM 开始的数据区和 TOS 开始的数据区, 这两个数据区分别以 0 作为结束标志.FROM 数据区记录所有航班的起点城市;TOS 数据区记录所有的航班的终点城市.下面的子程序 SERT 查找甲城市到乙城市是否有直接航班,若有则返回甲城号;若没有,则查找经过中间某一城市到达乙城的城市号返回;若没有这样的中间城市,则返回 0.调用该子程序时,GR0.GR1 分别存放甲.乙城市的编号.返回时,GR0 中存放结果.

[程序]

```
标号    指令码      操作数
SERT    START
        ST          GR0, X
        CALL        DIRT
        CPL         GR0, X
        JZE         RT
        LEA         GR2, 0
L1      LD          GR0, FROM, GR2
        SLL         GR0, 0
        JZE         RT
        CPL         GR0, X
        JZE         L3
L2      LEA         GR2, 1, GR2
        JMP         L1
L3      LD          GR0, TOS, GR2
        CALL        DIRT
        CPL         GR0, TOS, GR2
        JNZ         L2
RT      RET
DIRT    ST          GR0, SAVE
        PUSH        0, GR2
        LEA         GR2, 0
L4      LD          GR0, FROM, GR2
        SLL         GR0, 0
        JZE         L7
        CPL         GR0, SAVE
        JZE         L6
L5      LEA         GR2, 1, GR2
        JMP         L4
```

```
L6      CPL         GR1, TOS, GR2
        JNZ         L5
POP     POP         GR2
        RET
FROM    DS          80
TOS     DS          80
X       DS          1
SAVE    DS          1
        END
```

[分析]该程序为数组元素查找的一个应用，程序中调用 DIRT 子程序查找（GR0）、（GR1）两城市是否有直接航班。其方法为从 FROM 数据区的首单元开始查找其值为（GR0）的单元；若有，则查找 TOS 数据区的对应单元的值是否为（GR1），若是则说明（GR0）、（GR1）两城市有直接航班，否则继续从 FROM 数据区查找其值为（GR0）的下一单元，直到找到或找到值为 0 的单元。若（GR0）、（GR1）两城市没有直接航班，则查找（GR0）到达的某一中间城市。然后再查找中间城市到（GR2）是否有直接航班，若有则该中间城市返回，程序中各语句的具体分析说明如下：

标 号	指 令 码	操 作 数	说 明
SERT	START		
	ST	GR0, X	(GR0)→X
	CALL	DIRT	判（GR0），（GR1）间有没有直接航班
	CPL	GR0, X	(GR0)=(X),有直接航班，转 RT
	JZE	RT	
	LEA	GR2, 0	0→GR2,FROM 数据区指针赋初值
L1	LD	GR0, FROM, GR2	从 FROM 数据区取一城市编号送 GR0
	SLL	GR0, 0	（GR0)=0,结束，转 RT
	JZE	RT	
	CPL	GR0, X	（GR0)=甲城市编号转 L3
	JZE	L3	
L2	LEA	GR2,1,GR2	调整 FROM 数据区指针转 L1
	JMP	L1	
L3	LK	GR0,TOS,GR2	取甲城市到达的某中间城市编号送 GR0
	CALL	DIRT	判中间城市到乙城市有没有直接航班
	CPL	GR0,TOS,GR2	没有转 L2

	JNZ	L2	
RT	RET		
DIRT	ST	GR0，SAVE	（GR0)→SAVE
	PUSH	0，GR2	（GR2）压栈保存
	LEA	GR2，0	FROM 数据区指针赋初值，0→GR2
L4	LD	GR0，FROM，GR2	从 FORM 数据区取一城市编号送 GR0
	SLL	GR0，0	城市编号为 0，转 L7
	JZE	L7	
	CPL	GR0，SAVE	（GR0）=（SAVE）转 L6
	JZE	L6	
L5	LEA	GR2，1，GR2	调整 FROM 数据区指针转 L4
	JMP	L4	
L6	CPL	GR1，TOS，GR2	TOS 数据区对应城市编号≠乙城市编号
	JNZ	L5	转 L5
L7	POP	GR2	
	RET		
FORM	DS	80	
TOS	DS	80	
X	DS	1	
SAVE	DS	1	
	END		

上面讨论了一维数组的两个应用，下面主要讨论二维数组的存储结构及其应用。

1. 矩阵的存储形式

矩阵 A（m，n）=

```
a11   a12   a13   …   a1n
a21   a22   a23   …   a2n
       …          …
am1   am2   am3   …   amn
```

它的存储形式可以按矩阵的列序存储，也可以按矩阵的行序存储。

（1） 按列序存储的形式为：元素 a11、a21、a31、…、am1、a12、a22、a32、…、am2、a1n、a2n、a3n、…、amn 依次存于从某单元开始的存储区域。即先存放第 1 列的各元素，然后再存放第 2 列，最后存放第 n 列。

若一个元素占一个单元，a11 元素存于 A 单元，则元素 aij 在内存的存储地址为：

adrij=A(j-1)m+i-1

（2） 按行序存储的形式为：元素 a11、a12、a13、…、a1n、a21、a22、a23、…、a2n、…、am1、am2、am3、…、amn 依次存于从某单元（如 A 单元）开始存储区域。即先存放第 1 列的各元素，接着存放第 2 列，如此去，最存放 10 第 m 行。

若一个元素占一个单元，a11 元素存于 A 单元，则元素 aij 在内存的储地址为：

adrij=A+(i-1)n+j-1

[例 7.13]矩阵转置

下面一程序是将矩阵 A（10，10）在原数据区进行转置。A（10，10）的转置矩阵 A’（10，10）和 A（10，10）的关系为：

aij=a’ji (1≤i;j≤10)

A(10，10)按列序存于 A 单元开始的数据区内。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号要写。

[程序]

标 号	指 令 码	操 作 数	说 明
PRG	START		
	LEA	GR1,0	设置外循环初值
	{ ① }		设置内循环初值
LOOP2	LEA	GR3,0,GR1	10（j-1）+i-1
	SLA	GR3,2	
	ST	GR1,WRK	
	{ ② }		
	SLA	GR3,1	
	ST	GR2,WRK	
	{ ③ }		
	LEA	GR4,0,GR2	10(i-1)+j-1
	SLA	GR4,2	
	ST	GR2,WRK	
	{ ④ }		
	SLA	GR4,1	
	ST	GR1,WRK	
	ADD	GR4,WRK	
	LD	GR0,A,GR3	对角元素 aij,aji 交换
	ST	GR0,WRK	

	LD	GR0,A,GR4	
	{ ⑤ }		
	{ ⑥ }		
	ST	GR0,A,GR4	内循环调整
	LEA	GR2,1,GR2	内循环控制
	CPA	GR2,C10	
	JMI	LOOP2	外循环调整
	LEA	GR1,1,GR1	外循环控制
	CPA	GR1,C9	
	JMI	LOOP1	
	EXIT		
A	DS	100	
WRK	DS	1	
C10	DC	10	
C9	DC	9	
	END		

[分析]方阵的转置操作，就是将其对角元素交换的操作，因此对方阵中的下三角中各元素与对应的对角元素交换即可得转置矩阵。即：

A（10，10）=

```
 a11   a12   a13 … a1 10
 a21   a22   a23 … a2 10
        …         …
a10 1 a10 2 a10 3…a10 10
```

＝>转置

```
 a11   a21   a31 … a10 1
 a12   a22   a32 … a10 2
        …     …
a1 10 a2 10 a3 10…a10 10
```

程序处理时对下三角中各元素逐列处理。首先把元素 a21、a31、…、a10 1 与其对应的对角元素进行交换。再对 a32、a42、…、a10 2 与其对应的对角元素交换，如此反复。

下三角元素 aij(i＞j;1≤i,j≤10)的存储地址为：

adrij=A+10(j-1)+i-1

aij 的对角元素 aji 的存储地址为：

adrji=A+10(i-1)+j-1

程序中 GR1 为外循环记数器，记录 j-1，GR1 的值为 0。GR2 为内循环计数器，记录 i-1。GR3 存放元素 aji 所在单元的相对地址 10（j-1）+i-1。GR4 存放元素 aji 所在单元的相对地址 10（i-1）+j-1。程序处理框如图 7.20 所示。

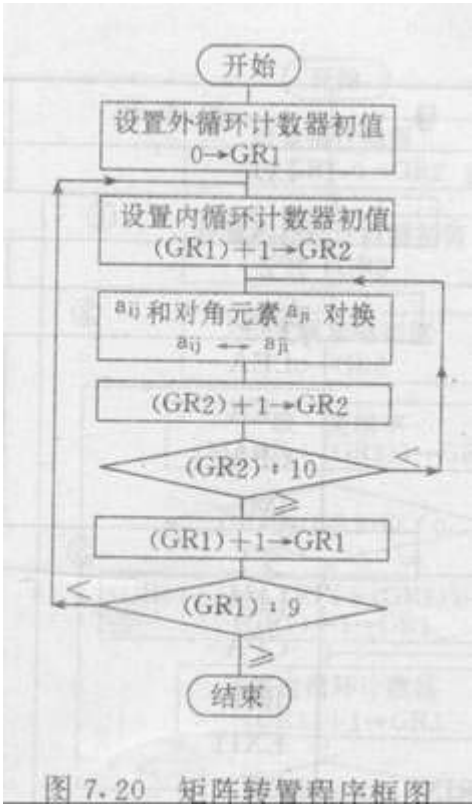


图 7.20 矩阵转置程序框图

[答案简析]
① 空应填指令：LOOP1 LEA GR2，1，GR1
设置内循环计数器 GR2 的初值。

② 空应填指令：ADD GR3，WRK
在 GR3 中求得 10（j-1）。

③空应填指令：ADD GR3，WRK
在 GR3 中求得 10(j-1)+i-1 作为 aij 元素的相对址。

④ 空应填指令：ADD GR4，WRK
类似于②空，在 GR4 中求 10(i-1)。

⑤、⑥空应填指令：ST GR0，A，GR3
LD GR0，WRK
实现 GR3，GR4 所指的对角元素交换。

2. 稀疏矩阵的压缩存储

如果一个矩阵中大多元素都为零元素，少数元素为非零元素，这样矩阵为稀疏矩阵。如果将矩阵中的所有元素都存于内存，显然会造成大多数空间的浪费。经常采用的一种存储形式是将非零元素和非零元素在矩阵中的位置存储下来。如一个稀疏矩阵中有 7 个非零元素，在内存中存储这 7 个非零元素，另外再保存这 7 个元素在矩阵中的位置。经常采用（16，16）矩阵中各元素在矩阵中的位置。如一个矩 A（16，16），可用 16 个逻辑字来一示 A（16，16）矩中元素为零还是为非零，第一个逻辑字对应于矩阵的第一行，第二个逻辑字对应于该行的某一列上的元素，若该位为 1 则对应元素非零，若该伴为零则对应元素为零。第 0 位对应于第一列，第 1 位对应于第 2 列，等，第 15 位对应于第 16 列。如第 3 个逻辑字上的第 4 位对应于元素 a35，如果该位为 0,则 a35 为零元素，如果该位为 1,则 a35 为非零元素。

[例 7.14]稀疏矩阵按列求和。

矩阵 A（16，20），其元素按列为序存放于 MATRIXAGA 开始的连续单元。由于 A（16，20）中有大量零元素，为了节省内存，凡非零元素才存储，，否则不存储。其元素是否为零，用 20 个逻辑常数记录。一个逻辑字对应于矩阵中的一列，如第 j 个逻辑常数对应于矩阵中第 j 列。第 j 个逻辑常数的第 0 位对应于元素 a1j,第一位对应于元素 a2j,…,第 15 位对应于元素 a16j。逻辑字中某位为 0 表示对应元素为零元素，某位为 1 表示对应元素为非零元素。逻辑常数存于 MARK 开始的单元。

下面一程序求矩阵中各列元素之各，且依次送到 ROWA 开始的 20 个单元中。

[问题]填充下成程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数	说 明
PRG	START		
	LEA	GR1,0	外循环设置初值
	LEA	GR2,0	
LOOP1	LEA	GR0,0	内循环设置初值
	LEA	GR3,0	
LOOP2	LD	GR3,MARK,GR2	内循环体
	{ ① }		
	JPZ	L1	
	{ ② }		
	LEA	GR1,1,GR1	
L1	LEA	GR3,1,GR3	内循环调整
	CPA	GR3,C16	内循环控制
	JMI	LOOP2	
	{ ③ }		

	LEA	GR2,1,GR2	外循环调整
	CPA	GR2,C20	外循环控制
	JMI	LOOP1	
	EXIT		
MATRIX	DS	320	
MARK	DS	20	
ROWS	DS	20	
C16	DC	16	
C20	DC	20	
	END		

[分析]按 MARK 单元开始的逻辑字求矩阵 A 的各列元素之和，在此过程中首先从高位到低检测逻辑字的各位是否为 1，若是，则累加对应元素，否则检测下一位，如此反复。程序中 GR0 作为累加器求各列元素之和，GR1 为 MATRIK 数据区指针，初值为 0；GR2 为 MARK 数据区指针，初为 0；GR3 为内循环计数器；GR4 存放对应的逻辑字。该程序的结构是二重循环结构，程序处理框图如图 7-21 所示。

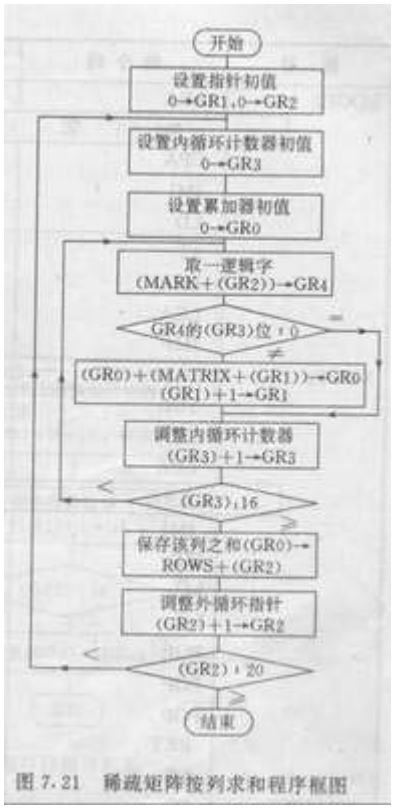


图 7.21 稀疏矩阵按列求和程序框图

[答案简析]

① 空应填指令：SLL GR4 ，0，GR3
前一条指令 LOOP2LD GR4，MARK，GR2 将对应列的逻辑字送入 GR4。要判断该列上的对应元素是否非零，则要检测逻辑字的对应位是否为 1，就必须将逻辑字的对应位移到最高位。

② 空应填指令：ADD GR0，MATRIX，GR1
对应元素非零，进行累加。

③ 空应填指令：ST GR0 ，ROWS，GR2 将某列元素的和存于 ROWS 数据区的对应单元。

3、 三角矩阵的压缩存储

（1） 上三角为零的稀疏矩阵

矩阵 A（4，4）=

1	0	0	0
3	7	0	0
5	1	9	0
8	4	6	3

即为上三角为零的稀疏矩阵，该矩阵的上三角均为零。如果将矩阵 A 从内存的 A 单元开始按行存储且零元素不存，则一非零元素 aij(i≥j)在内存单元的地址为（设一个元素占一个单元）

adrij=A+(i(i-1))/2+j-1

如，a43 相对于 A 单元的存储地址为：

(4(4-1))/2+3-1=8

（2） 下三角为零的稀疏矩阵：

矩阵 A(4,4)=

5	7	6	8
0	10	4	9
0	0	3	7
0	0	0	6

即为下三角为零的稀疏矩阵，该矩阵的下三角均为零。下三角为零的矩阵 A（N，N）按行压缩存储后，a11 元素存于 A 单元，一个元素占一个单元，非零元素 aij(j≥i)的存储地址为：

adrij=A+N(i-1)-(i(i-1))/2+j-1

如，在 A（4，4）矩阵中，元素 a23 相对于 A 单元的存储地址为：

4（2-1）-2（2-1）/2+3-1=5

[例 7。15]上三角为零的矩阵压缩存储。

一个上三角为零的矩阵 A（n，n），已按行序存放在 GR3 所指单元开始的存储区中，每个元素占一个单元，存放时未经压缩。下面的子程序将其变换成按行压缩的存储的形式。调用子程序时，阶数 n 存于 GR0，A（n，n）矩阵存放的首地址存于 GR3。

[问题]填充下面程序中的 ，完成程序，不必要的标号不要写

[程序]

标号	指令码	操作数	说明
TRIQUE	START		
	ST	GR0,N	保存寄存器的值
	PUSH	0,GR1	
	PUSH	0,GR2	
	PUSH	0,GR3	
	ST	GR4,SPSAVE	
	{ ① }		设置外循环初值
	LEA	GR1,1	
LOOP1	LEA	GR2,0	设置内循环初值
LOOP2	LEA	GR2,1,GR2	内循环调整
	{ ② }		内循环控制
	CPA	GR1,WORK	
	JMI	NEXT2	
	LD	GR0,0,GR4	
	ST	GR0,0,GR3	
	LEA	GR3,1,GR3	
	LEA	GR4,1,GR4	
	JMP	LOOP2	
NEXT2	{ ③ }		
	SUB	GR4,WORK	
	LEA	GR4,1,GR4	
	LEA	GR1,1,GR1	外循环调整
	{ ④ }		外循环控制
	JMI	LOOP1	
	JZE	LOOP1	
	LD	GR0,N	恢复寄存器的值
	LD	GR4,SPSAVE	
	POP	GR3	
	POP	GR2	

	POP	GR1	
	RET		
N	DS	1	
SPSAVE	DS	1	
WORK	DS	1	
	EDN		

[分析]由于存放矩阵 A（n，n）的数据区是未压缩的，因此数据区中存放了许多零元素。第一行有 n-1 个零元素，第二行有 n-2 个零元素，等等。在进行压缩过程中都要跳过这些零元素。将各行的非零元素依次存放。在压缩过程中采用了两个指针，一是未压缩数据区的指针 GR4，另一是压缩数据区的指针 GR3。按行压缩，先将第一行的非零元素传入压缩数据区，然后指针 GR4 跳过零元素。再将第二行中的非零元素传入压缩区，如此下去，一直到压缩完最后一行。该程序的结构是二重循环结构。GR2 为矩阵列计数器，GR1 为行计数器，程序处理框图如图。7.22 所示。

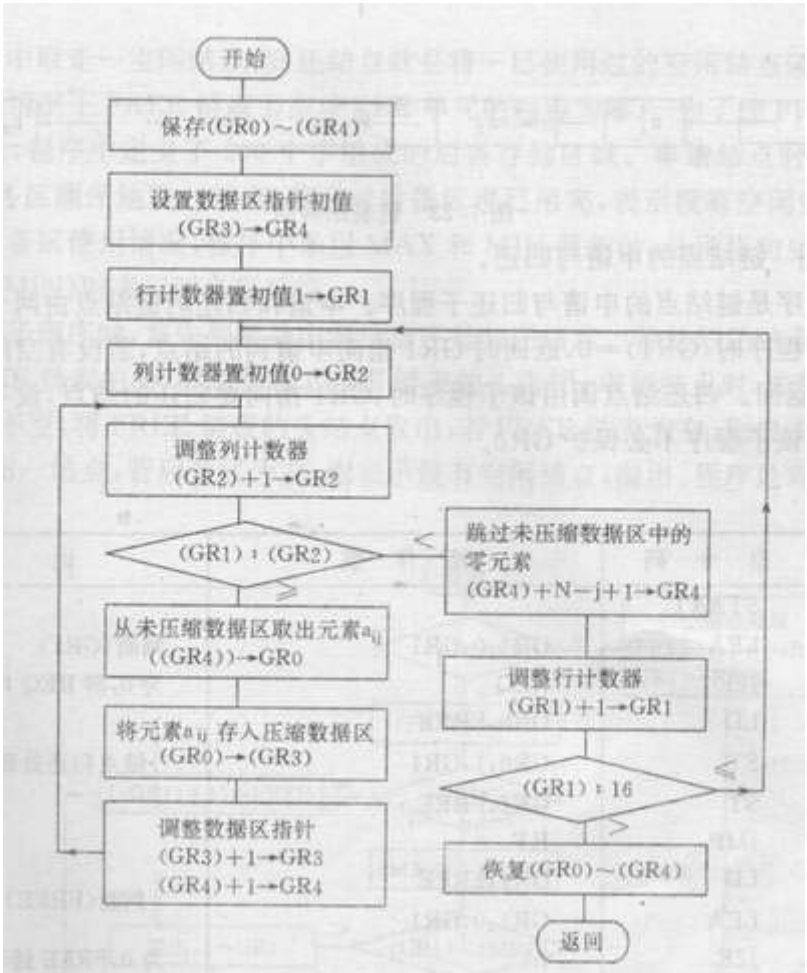


图 7.22 上三角为零的矩阵压缩存储程序框图

[答案简析]

① 空应填指令：LEA GR4 ，0，GR3
因为 GR4 为未压缩数据区指针、GR3 为压缩数据区指针，在初始情况下它们的值相同，都指向数据区首单元。

② 空应填指令：ST GR2，WORK
因下一条指令为 CPA GR1 ，WORK；在 GR1 中记录的是矩阵的行。上三角为零的矩阵中，列号大于行号的元素都为零元素，此时要跳过这些零元素进行压缩，该空指令将列号存于 WORK 单元，以便下面进行行号和列号比较。

③ 空应填指令：ADD GR4 N
要跑过未压缩数据区中的零元素，GR4 指针要进行调整 （GR4）+n-j+1→GR4。下面两条指令实现（GR4）-j+1=GR4。因此该空所填指令实现：（GR4）+n→GR4。

④ 空应填指令：CPA GR1，N 或 CPL GR1，N
因为下面一条指令为 JMI LOOP1 进行循环控制。

7．3.2 链表

与数组不同，链表中的元素不一定连续地存放在内存区中，各元素之间的关系是通过链指针连系的。如图 7.23 所示，用链表构造了数据元素 a1、a2、…an 之间的顺序关系。



图 7.23 链表结构图

在链表中每个结点由两个部分组成，一是数据项，如 a1、a2、…an。另一是指针项，指出下一结点所在的位置。最后一个结点的指针项是空指针 0。此外还有一个头指针 FIEST,FIEST 单元的值指出链表中头一个结点的位置。与数组相比，每个结点增加了为设置指针而消耗的额外空间。但这种方法避免了在数组中用连续单元存储数组元素的缺点，在进行插入或删除操作时，不需要移动元素来腾出空间或填补空缺。在链表中结点的插入和删除只要对链指针进行修改即可。

[例 7.16]链结点的申请与归还。

下面的程序是链结点的申请与归还子程序。申请和归还的链结点由两个字组成。申请结点调用该子程序时（GR1）=0，返回 GR1 指向申请到的结点，若没有空闲结点，即溢出，则（GR1）=0 返回，归还结点调用该子程序时，GR1 指向要归还的结点，设不会归还首址为零的结点，该子程序不必程序 GR0。

[程序]

标 号	指 令 码	操 作 数	说 明
REQL	START		
	LEA	GR1,0,GR1	判断(GR1)

	JZE	REQ	为 0，转 REQ 申请结点
REL	LD	GR0,FREE	结点归还处理
	ST	GR0,1,GR1	
	ST	GR1,FREE	
	JMP	RT	
REQ	LD	GR1,FREE	判断（FREE）
	LEA	GR1,0,GR1	
	JZE	L1	为 0，FREE 链表为空，转 L1
	LD	GR0,1,GR1	FREE 链表不空，取出头结点
	ST	GR0,FREE	
	JMP	RT	
L1	LD	GR1,MAX	后备区空吗？
	LEA	GR1,2,GR1	
	CPL	GR1,MIN	
	JZE	L2	不空，转 L2
	JMI	L2	
	LEA	GR1,0	空，溢出！
	JMP	RT	
L2	ST	GR1,MAX	从后备区取一结点
	LEA	GR1,-2,GR1	
RT	RET		
L0	DS	200	
MIN	DC	MIN	
MAX	DC	L0	
FREE	DC	0	
	END		

[分析]为了实现链结点的申请与归还，程序中建立了空闲结点链表 FREE，该链表中的所有结点都是空闲的。申请结点和归还结点就是对 FREE 链表的操作。申请结点就是从 FREE 链表中取走一空闲结点；归还结点就是将一已使用过的空闲还给 FREE 链表。

在初始情况下 FREE 链表为空（FREE 单元的内容为零）。为了使 FREE 队列为空时能申请到结点，程序中定义了 200 个字组成的后备在座区。申请到结点时，若 FREE 队列为空，则到后备区顺序地取一结点，若此时后备区也已用完，表示没有空闲结点，称之为溢出。为了记录后备区使用情况，程序中采用 MAX 和 MIN 两指针。分别指向备区的头和尾，当（MAX）≥（MIN）时表示后备区已空。

进入该子程序时，首先判断是申请结点还是归还结点。若是归还结点，则将归还的结点链入 FREE 链表的头，然后修改 FREE 链表的指针。申请结点时，先检查 FREE 链表是否为空。若不空。将 FREE 链表的头结点取出；若 FREE 链表为空，

则检查后备区；若后备区不空，则取出一结点；若后备区为空，则表示没有空闲结点，溢出。程序处理框图如图 7.24 所示。

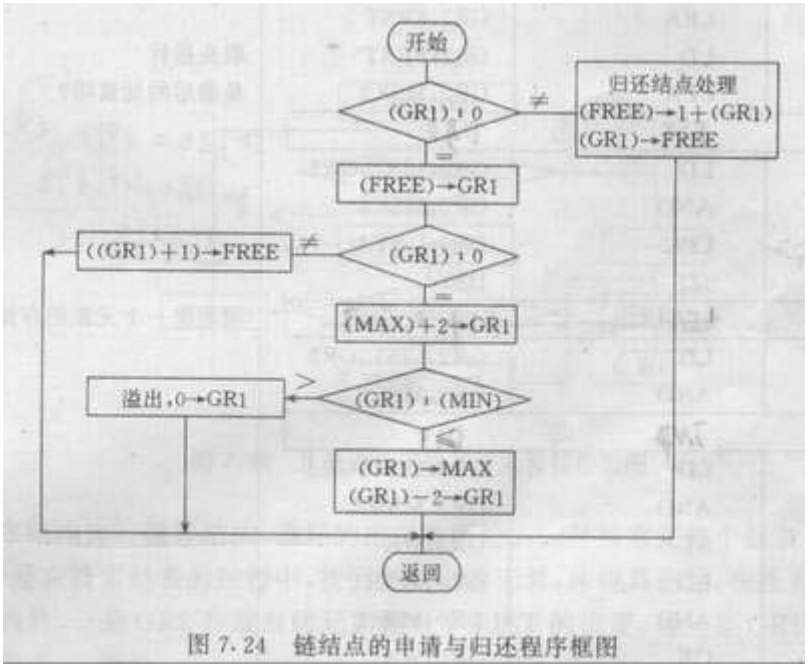


图 7.24 链结点的申请与归还程序框图

[例 7.17]链表删除。

一子程序的功能是从链表中删除与指定数据具有相同值的元素。

（1）、链表放置在以 LIST 地址开始的 200 个连续单元中。

（2）、链表的各元素的构成如下所示，每个链表元素占一个存储字。

0 7	8 15
数 据 部	指 针 部

（3）、指针部是放置了相对于 LIST 地址的相对位移量，链表的最后元素的指针部内放置为（FF）16。

（4）链表的头指针（相对位置）放置在 FRST 地址中，但若链中无元素，则 FRST 存放（00FF）16。

（5）要删除元素的数据部值存放在 GR1 的 8-15 位，这样传递给程序。

（6）链表中数据部具有同一值的元素不多于 1 个。

（7）若在链表中找到同一值的元素，将 GR3 设置为 0；若不存在，则将 GR3 设置为-1，返回主程序。

[问题]填充下面程序中的确良 ，完成程序，不必要的标号不要写。

[程序]

```
标号   指令码        操作数        说明
DELE   START
       PUSH         0，GR1
       PUSH         0，GR2
       SLL          GR2，8
       ST           GR1，DATA
       LEA          GR1，FRST
       LD           GR2，FRST     取头指针
D10    CPA          GR2，MSK2     是最后的元素吗？
                  { ① }
       LD           GR0，LIST，
                    GR2
       AND          GR0，MSK1
       CPL          GR0，DATA
       JZE          D20
                  { ② }           设定前一个元素的存
       LD           GR2，LIST，   储地址
                    GR2
       AND          GR2，MSK2
                  { ③ }
D20    LD           GR3，LIST，
                    GR2
       AND          GR3，MSK2
       ST           GR3，WK
       LD           GR0，0，GR1
       AND          GR0，MSK1
       OR           GR0，WK
       ST           GR0，0，GR1
       LEA          GR3，0
                  { ④ }
D30    LEA          GR3，-1
D40    POP          GR2
       POP          GR1
       RET
DATA   DS           1
WK     DS           1
MSK1   DC           #FF00
MSK2   DC           #00FF
FRST   DS           1
LIST   DS           200
       END
```

[分析]阅读程序说明，可知该程序中所使用的数据结构如图 7.25 所示。



图 7.25 链表删除中使用的数据结构

从图 7.25 可以看出，处理过程中所使用的数据结构为链表结构，阅读程序可知程序处理框图如图 7.26 所示。

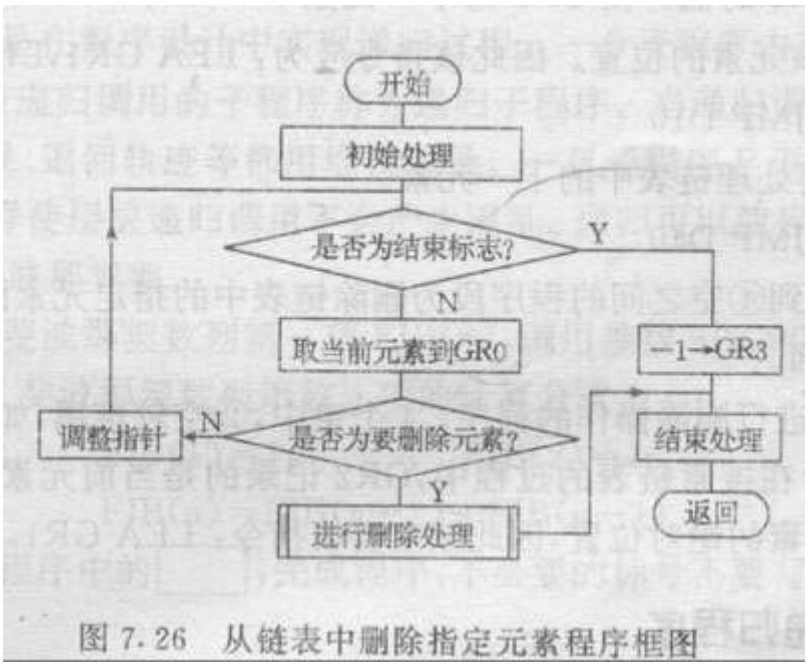


图 7.26 从链表中删除指定元素程序框图

整个程序结构为一循环结构，循环的出口有两处，一是搜查完整个链表，未找到要删除的元素；另一是在搜索链表的过程中，找到要删除的元素，并将其删除。在搜索链表的过程中使用了两个指针，一是 GR2 存放当前元素相对于 LIST 的位置 ；另一是 GR1 存放上一元素的位置。如图 7.27 所示。



图 7.27 链表搜索过程中 GR1、GR2 的作用

图 7.26 的框图中，进行删除处理框的细化框图如图 7,28 所示。

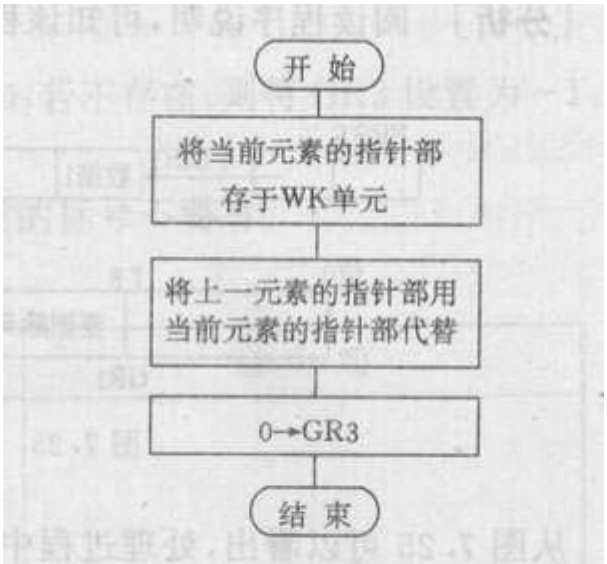


图 7.28 进行删除处理框的细化框图

程序在处理过程中使用了两个逻辑常数，MSK1（#FF00）、MSK2（#00FF）。MSK1 用于析取链表元素的数据部，MSK2 用于析取链表元素的指针部。程序中的 DATA 单元用于保存要删除的指定元素的数据值，经过以上分析结合程序框图和数据结构再阅读程序就可以完成程序中的空。

[答案简析]

① 空应填指令：JZE D30
因为上一条指令为 CPA GR2，MSK2 判断 GR2 的值是否为结束标志（#00FF），若是，说明已搜索完整个链表，转 D30 返回。

② 空应填指令：LEA GR1，LIST，GR2
因为上两条指令为：CPL GR0，DATA 和 JZE D20 判断当前元素是否为要删除的元素。若是，则转标号 D20 处进行删除处理；若不是，则进行调整指针处理，即调整 GR1 和 GR2 的值。使 GR2 为下一元素的相对位置，GR1 为该元素的位置。因此该指令应为：LEA GR1， LIST，GR2。

③ 空应填指令：JMP D10
因为调整指针后，则要处理链表中的下一元素。

④ 从标号 D20 开始到④空之间的程序段为删除链表中的指定元素的处理部分。进行删除处理后，则转 D40 返回。

该程序是对链表进行删除操作的程序。4 个空中，②空较难填，如填写指令：LEA GR1，0，GR2 就错了。因为在搜索链表的过程中，GR2 记录的是当前元素相对于

LIST 的位置，GR1 记录的是上一元素的绝对位置，因此该空要填指令：LEA GR1 ，LIST， GR2。

7.3．3 栈与递归程序

所谓堆栈，就是一个按照后进先出原则组织的一段内存区域，在这段内存区中，有一个始终指向栈顶元素的指针，称为栈称针 SP。在初始情况下栈为空。栈指针指向栈底单元的下一单元。如图 7.1 所示。

CASL 程序中，用户可以自己定义栈区，栈区要在主程序中定义。如，下面是一个定义由 80 个单元组成的栈区的程序：

标号	指令码	操作数	说明
	START		
	LEA	GR4,STACK	主程序中给栈赋初值
	…		
	DS	80	在程序数据区定义 80 个单
STACK	DS	0	元的栈区
	END		定义标号 STACK

如果用户程序中未定义栈区，则使用系统栈区。栈指针 SP 和 GR4 兼作，同时，GR4 又可作一般的通用寄存器。这样，给掌握 CASL 中的堆栈和 CASL 程序设计带来一定的复杂性。主要表现在以下两点：

（1）一般情况下程序中不要使用 GR4，若要使用，先保护 GR4 的值，用完后再恢复。如果是一个简单的 CASL 程序，程序中未出现堆栈指令（POP，PUSH，CALL，REL）则 GR4 可作为通用寄存器或变址寄存使用，不受任何限制。

但是，程序中通常要使用到堆栈指令，因此要尽量避免使用 GR4。如果要使用 GR4。应在程序的数据区定义一栈指针单元，将 GR4 的值保存于该单元，然后才使用 GR4。用完后，再将保存于该单元的值恢复到 GR4。

（2）利用 GR4 兼作栈指针和变址寄存器，可以设计出较简洁的程序。利用 GR4 作为变址寄存器，非堆栈指令也可以使用栈区的数据。

例如，指令：

```
PUSH 0，GR2
ADD GR1，0，GR4
```

实现（GR2）+（GR1）→GR1

栈的一个重要作用是在程序设计中实现递归过程。一个子程序内有调用本子程序的语句时，就形成了递归。有递归调用的子程序称为递归子程序。当递归调用子程序 P 时，递归子程序的参数、中间结果、返回， 轨迹等都用堆栈记录。一旦

子程序 P 返回时，则将相应的现场记录从栈中恢复。这样使用递归调用不会产生混乱。递归可以使程序的结构得到简化。

[例 7.18]计算斐波那契数。

下面一子程序计算斐波那契列数第 n 项 FIB(n)，调用参数 n（n>0）的整数存于 GR1，结果 FIB（n）存于 GR2。斐波那契数列第数 n 项的计算公式为：

FIB（1）=1， FIB（2）=1

FIB（n）=FIB(n-2)+FIB(n-1)

[问题]填充下面程序中的 ，完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数	说 明
FIB	START		
	CPA	GR1,C1	
	JZE	BASIS	
	CPA	GR1,C2	
	JZE	BASIS	
	PUSH	0,GR1	
	{ ① }		
	CALL	FIB	求 FIB(n-2)
	POP	GR1	
	{ ② }		保存 FIB(n-2)
	LEA	GR1,-1,GR1	
	CALL	FIB	求 FIB(n-1)
	POP	GR1	
	{ ③ }		
	ADD	GR2,WRK	求 FIB(n-2)+FIB(n-1)
	JMP	BACK	
BASIS	LEA	GR2,1	
BACK	RET		
C1	DC	1	
C2	DC	2	
WRK	DS	1	
	END		

[分析]计算 FIB（n），可归结为求解以下几个问题：

（1） 计算 FIB（n-2）

（2） 计算 FIB（n-1）

（3） 将（1）和（2）所得结果相加；

（4） 边界条件为 FIB（1）=1，FIB（2）=1。

求解（1）和（2）就是进行递归调用。求（1）和（2）的过程和求 FIB（n）的过程一样，所不同的是调用参数的不同。在进行递归调用前，要保存是调用参数 GR1 的值，递归调用后要恢复 GR1 的值。在解决（1）的递归调用后，，FIB（n-2）的结果存于 GR2。为了避免在求解（2）的过程中，将（1）的结果 FIB（n-2）冲掉，因此在进行（2）的递归调用前，要将 FIB（n-2）的结果压栈保存，递归调用后恢复其值到 WRK 单元，然后再和 GR2 中的 FIB（n-1）相加求得 FIB（n）. 程序中 GR1 存放调用参数 n，GR2 存放结果 FIB（n）。程序处理框图如图 7.29 所示。



图 7.29 递归求 FIB(n)程序框图

[答案简析]

① 空应填指令：LEA GR1，-2 ，GR1 或 SUB GR1，C2

因为下一条指令 CALL FIB 进行递归调用，因此 在递归调用前要修改调用参数。

② 空应填指令：PUCH 0，GR2

因下面的两条指令为求 FIB（n-1）的递归调用，为了避免 FIB（n-2）的值被冲掉，因此在求 FIB（n-1）前将 GR2 的值压栈保存。

③ 空应填指令 ST GR1，WRK

上一条指令 POP GR1 将 FIB(n-2)的值弹出到 GR1,为了求得 FIB(n-2)+FIB(n-1)，将 FIB（n-2）的值存入 WRK 单元。

该程序利用 WRK 工作单元求 FIB（n-2）+FIB（n-1），也可以利用 GR4 兼作栈指针和变址寄存器，求 FIB（n-2）+FIB(n-1)。将程序中指令序列：

```
POP             GR1
```

③

```
ADD             GR2，WRK
换成：
ADD             GR2，0，GR4
POP             GR1
```

该程序也同样正确。

7.3。4 二叉树

与链表不同,二叉树的每个结点包含两个指针部分,分别指向其左子树与右子树。二叉树的结点由三部分组成：数据部分，左子树指针，左子树指针：

数据部分
左子树部分
右子树部分

在 CASL 中一般约定，若无结点与对应的指针相连，则该指针值为 0。若二叉树每个结点左指针连接的结点比本结点小，右指针连接的结点比本结点大。则称为有序二叉树，图 7.3 所表示的二叉树即为有序二叉树，ROOT 单元存放二叉树根结点的地址。

[例 7.19]有序二叉树插入。

一有序二叉树的结点由三个单元组成，第一单元存放数据，第二单元为左结点指针，第三单元为右结点指针，若结点的指针部分无子结点与之相连。则该指针值为 0。ROOT 单元存放根结点的地址。如图 7.30 所示。

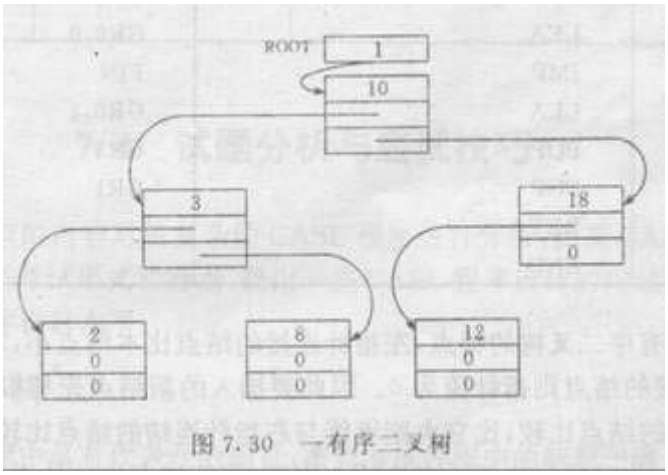


图 7.30 一有序二叉树

下面一子程序将给定的一新结点插入到二叉树中。

（1）在调用该子程序时，GR1 中存放 ROOT 单元的地址：GR2 中存放新结点的地址：此时新结点的第 1 个单元为数值，第 2、3 单元为 0。

（2）当新结点的数值与二叉树中有同值，则置（GR0=1），返回；若新结点在二叉树中不存在，则插入。

（3）若 ROOT 单元的值为 0，则为空二叉树。

（4）子程序返回时 GR1～GR3 内容保持不变。

[程序]

标 号	指 令 码	操 作 数
BTREE	START	
	PUSH	0,GR1
	PUSH	0,GR3
	LD	GR3,0,GR1
	LEA	GR3,0,GR3
	JZE	STORE
LOOP	LD	GR1,0,GR3
	CPA	GR1,0,GR2
	JMI	RIGHT
	JZE	ERROR
	LEA	GR1,1,GR3
	JMP	NEXT
RIGHT	LEA	GR1,2,GR3
NEXT	LD	GR3,0,GR1
	LEA	GR3,0,GR3

	JNZ	LOOP
STORE	ST	GR2,0,GR1
	LEA	GR0,0
	JMP	FIN
ERROR	LEA	GR0,1
FIN	POP	GR3
	POP	GR1
	RET	
	END	

[分析] 根据有序二叉树的特点，左指针连接的结点比本结点小，右指针连接的结点比本结点大，若无连接的结点则指针值为 0。因此要插入的新结点先和根结比较；比它小则继续与左指针连接的结点比较；比它大则继续与右指针连结的结点比较，直到被查询的结点的相应指针值为 0。此时，即找到新结点的插入位置，将新结点的地址送入该结点的相应指针部分。在程序中 GR2 指向要插入的新结点，GR3 指向二叉树中比较结点。初始情况下，GR3 指向根结点（取 ROOT 单元的值）。在比较过程中调整 GR3 的值，即可实现与左结点比较或与右结点比较。程序中各语句的具体分析说明如下：

标 号	指令码	操作数	说明
	LEA	GR3,0,GR3	(GR3)=0，转 STORE；为空二叉树，转 STORE
	JZE	STORE	
LOOP	LD	GR1,0,GR3	二叉树当前结点与新结点比较
	CPA	GR1,0,GR2	
	JMI	RIGHT	<,转 RIGHT;当前结点小于新结点，转 RIGHT
	JZE	ERROR	=，转 ERROR
	LEZ	GR1,1,GR3	>,GR1 指向二叉树当前结点的右指针部分
	JMP	NEXT	
RIGHT	LEA	GR1,2,GR3	GR1 指向二叉树当前结点的右指针部分
NEXT	LD	GR3,0,GR1	
	LEA	GR3,0,GR3	二叉树当前结点的指针单元不为 0,转 LOOP
	JNZ	LOOP	
STORE	ST	GR2,0,GR1	插入新结点
	LEA	GR0,0	0→GR0
	JMP	FIN	
ERROR	LEA	GR0,1	1→GR0
FIN	POP	GR3	恢复(GR3),(GR1)

	POP	GR1	
	RET		
	END		

7.4 试题分析与应试技巧

本节结合前面三节的内容对较复杂的 CASL 程序进行分析，提高 CASL 程序设计能力和分析能力。针对水平考试形式和特点。提出一些 CASL 程序设计的应试技巧，帮助读者提高应试能力和汇编程序设计水平。

7．4.1 试题分析

这里给出汇编试题中具有代表性的题目，按照考试过程中解题步骤，按步分析，读者可以体会到汇编试题的解题过程和解题方法，从而提高应试能力。

[例 7.20]字符串比较。

有一子程序，其功能是比较两个长度相同的字符串，将比较结果设置于标志寄存器 FR 中。

（1） 由主程序将两个字符串的首地址及长度以下表的形式传入子程序，该表的地址放在 GR3 中。

字符串 1 的首地址	字符串 2 的首地址	字符串长度

（2）该子程序从字符串 1、2 的头开始，每次对一个字符进行比较，直到指定的长度为止。比较途中，如比较出字符不等，则中止比较，将 FR 的值返回主程序。

（3）字符串长度在 1～255 之间。

（4）在子程序执行时，GR3 以外的寄存器的内容不保存也可以。

[问题]填充下面程序的{ } ，完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数	说 明
CHCOMP	START		
	LD	GR0,2,GR3	
	{ ① }		字符串 1 的最终地址+1
	ST	GR0,ENDADR	→ENDADR

	{ ② }		
	LD	GR2,1,GR3	字符串 2 的首地址→GR2
LOOP	LD	GR0,0,GR1	
	CPL	GR0,0,GR2	字符串 1 的字符：字符串 2 的字符
	{ ③ }		
	LEA	GR1,1,GR1	字符串 1 的指针加 1
	LEA	GR2,1,GR2	字符串 2 的指针加 1
	CPA	GR1,ENDADR	
	JNZ	LOOP	
LAST	RET		
ENDADR	DS	1	
	END		

[分析]阅读程序说明，从程序说明（1）可知，程序执行时的数据结构如图 7.31 所示。从程序说明（2）和程序中的注释可知，程序中使用的比较算法为：从串 1、串 2 的头开始，逐个字符进行比较。如果中间出现不相等的字符则中断比较，说明两字符串不等；否则继续比较，直到结束，两字符串相等。在处理过程中，GR1 作为串 1 的指针，GR2 作为串 2 的指针。程序处理框图如图 7.32 所示。

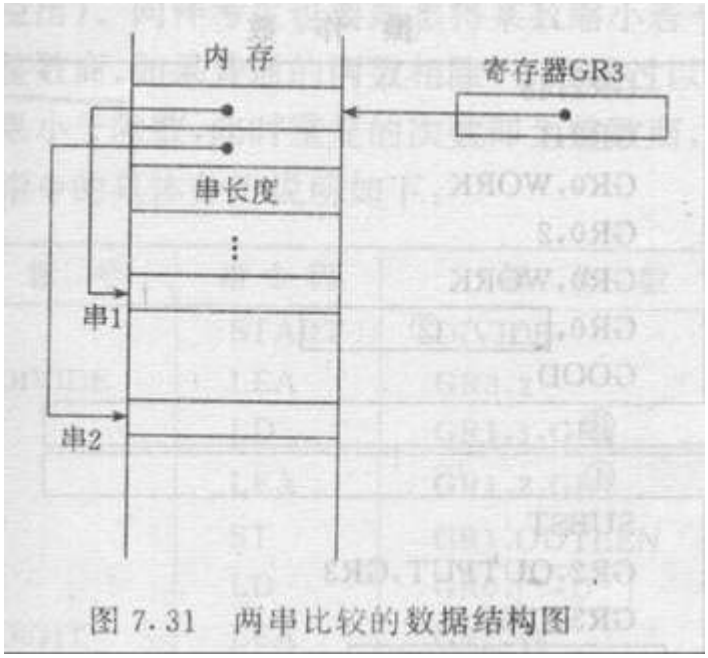
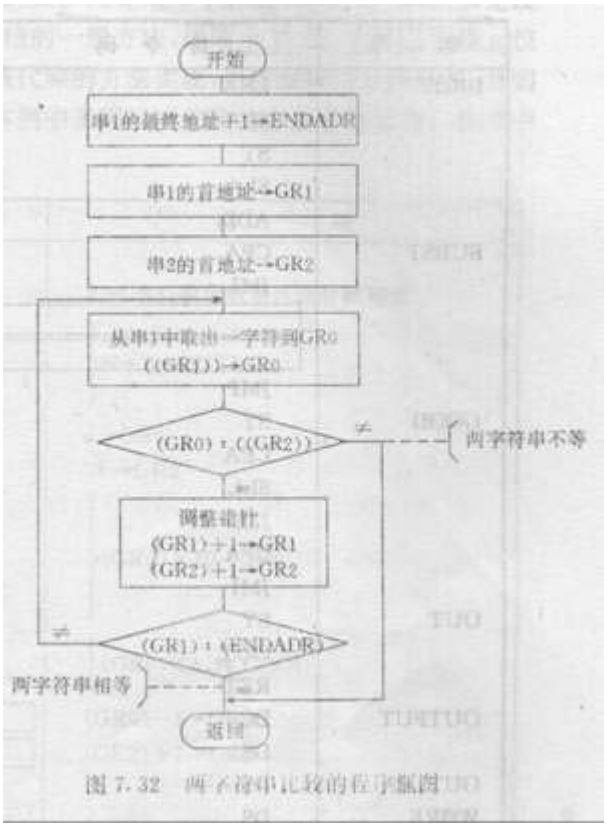


图 7.31 两串比较的数据结构图



从程序框图可以看出，当两字符串不等时，FR 的值为 10 或 00；当两字符串相等时，FR 的值为 01，结合程序框图和程序注释程序，可以得出以下的答案。

[答案简析]

① 空应填指令 ADD GR0，0，GR3
得到串 1 的终地址加 1

② 空应填指令：LD GR1， 0，GR3
将串 1 的首地址送 GR1。因为 GR1 作为串 1 的指针

③ 空应填指令：JNZ LAST

因为上一条指令为：CPL GR0，0GR2 实现串 1 的字符与串 2 的字符比较。如不等。则（FR）=10 或 00，转返回。

[小结]从此例可以看出，对水平考试中的汇编试题，一般的解题过程可以归纳成以下三个步骤：
（1）阅读程序说明，了解程序功能以及完成该功能的实现方法和数据结构（有的试题中未给出程序中完成功能的方法，则需要阅读程序了解完成功能的实现方法）

（2） 阅读程序，了解程序中所使用的寄存器、变量和数据区的作用。如本例中 GR1 为串 1 数据区指针，GR2 为串 2 数据区指针，GR0 存放串 1 中的字符等。进

一步了解程序中所使用的算法和数据结构。画出程序框图和数据结构图（这一步所花时间较多，在考试中受时间限制，可在草稿上画出一些草图）。

（3） 结合程序框图和数据结构，再读程序，填出各空应填的指令。

[例 7.21]子程序 DIVIDE 将真分数 a/b(0<a<b) 转换成 N 位小数（0<N ≤20），并输出结果。

子程序从栈中接受参数。参数的入栈顺序为：分子(a)、分母(b)和小数位数(N)。进入子程序时，已保证 0<a<b 且 o<N ≤20，子程序不再对参数作合法性检查。

由于需要将结果输出，所以子程序中计算出每一位小数后，直接用字符方式顺次存放在标号 OUTPUT 之后的存贮区域中，数字字符 0--9 的 ASCII 码依次是 48-57。

[问题]填充下面程序的{ } ，完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数
	START	DIVIDE
DIVIDE	LEA	GR3,2
	LD	GR1,1,GR4
	LEA	GR1,2,GR1
	ST	GR1,OUTLEN
	LD	GR0,{ ① }
DIGIT	LEA	GR2,48
	SLA	GR0,1
	ST	GR0,WORK
	SLA	GR0,2
	ADD	GR0,WORK
SUBST	CPA	GR0，{ ② }
	JMI	GOOD
		{ ③ }
		{ ④ }
	JMP	SUBST
GOOD	ST	GR2,OUTPUT,GR3
	LEA	GR3,1,GR3
	SLL	{ ⑤ }
	JZE	OUT
	CPA	GR3,OUTLEN
	JMI	DIGIT
OUT	ST	GR3,OUTLEN

	ORT	OUTPUT,OUTLEN
	RET	
OUTPUT	DC	’0’
	DS	20
OUTLEN	DS	1
WORK	DS	1
	END	

[分析]因主程序在调用 DIVIDE 前将参数按 a、b、N 的次序压栈，进入 DIVIDE 子程序后栈区的结构与内容为：

(GR4) +0	返回地址
+1	N
+2	b
+3	a

阅读程序可知，实现 a/b 的方法是将被除数 a 扩大 10 倍，然后求除以 b 的整数商，则得到第 1 位小数；对除以 b 的余数再扩大 10 倍作为被除数，再求除以 b 的整数商，则得到第 2 位小数。如此循环，直到余数为 0，或已求得 N 位小数。最后将求得的结果输出。下面的几组指令均实现将 GR0 的值扩大 10 倍，程序中采用第①组指令实现将被除数扩大 10 倍。

①	②	③
SLA GR0,1	ST GR0,WORK	ST GR0,WORK
ST GR0,WORK	SLA GR0,3	SLA GR0,2
SLA GR0,2	ADD GR0,WORK	ADD GR0,WORK
ADD GR0,WORK	ADD GR0,WORK	SLA GR0,1

考生应熟悉将某数扩大若干倍的一般方法，如扩大 2_n 倍，可通过左移 n 位实现（假定没有溢出）。同样考生也要熟悉将某数缩小若干倍的一般方法，如缩小 2_n 倍，可通过右移 n 位求整数商。如果普通的两数相除，一般通过以减代除的方法实现，用被除数重复减除数，直到结果小于除数，此时重复的次数即为整数商，本例中采用减代除的方法求整数商。程序中各指令的具体分析如下：

标 号	指 令 码	操 作 数	说 明
	START	DIVIDE	
DIVIDE	LEA	GR3,2	2→GR3；输出数据区指针赋初值
	LD	GR1,1,GR4	N+2→OUTLEN
	LEA	GR1,2,GR1	
	ST	GR1,OUTLEN	

	LD	GR0,{ ① }	a→GR0
DIGIT	LEA	GR2,48	'0'→GR2
	SLA	GR0,1	(GR0)×10→GR0
	ST	GR0,WORK	
	SLA	GR0,2	
	ADD	GR0,WORK	
SUBST	CPA	GR0, { ② }	(GR0)<b,转 GOOD
	JMI	GOOD	
		{ ③ }	(GR0)-b→GR0
		{ ④ }	(GR2)+1→GR2
	JMP	SUBST	
GOOD	ST	GR2,OUTPUT,GR3	(GR2)送输出数据区,调整GR3
	LEA	GR3,1,GR3	
	SLL	{ ⑤ }	(GR0)=0 转 OUT; 余数为 0 转输出
	JZE	OUT	
	CPA	GR3,OUTLEN	未求足 N 位小数, 转 DIGIT
	JMI	DIGIT	
OUT	ST	GR3,OUTLEN	将所求的小数输出
	ORT	OUTPUT,OUTLEN	
	RET		
OUTPUT	DC	'0'	
	DS	20	
OUTLEN	DS	1	
WORK	DS	1	
	END		

该程序中 GR3 作为 OUTPUT 输出数据区的指针，其初值为 2（因为 OUTPUT）数据区前两个单元为字符 0。GR2 作为减代除的循环计数器，GR0 存放被除数，以及整除后的余数。填①空时，根据①空下面 的几条指令实现（GR0）×10，可确定①空指令将被除数 a 送 GR0.。填②空时，已知 GR0 中为被除数，且该条指令实现以减代除的循环控制,故该指令实现被除数与除数比较,当被除数小于除数时，跳出循环。③空实现减法操作，④空循环计数，求得整数商。⑤空判余数（GR0）是否为 0，若为 0 则转输出结果。

[参考答案]

① 3，GR4

② 2，GR4

③ SUB GR0，2，GR4

④ LEA GR2，1，GR2（③和④可交换）

⑤ GR0，0

[例 7.22]下面的程序是按某种规律自动生成 16×16 单色点阵图形。点阵图形中每行 16 点的信息排列在一个存储字中，16×16 可以用连续存储字来表示。

程序中标号为 PTN 开始的 16 个存储字连续存放自动生成的 16×16 点图形，点阵图形的第一行作为已知数据给其余 15 行按下述规则自动生成：

（1） 每个存储字的第 0 位和第 15 位（即边界点）恒为 0。

（2） 每个存储字的第 n 位值（0<n <15）取决于前一个存储字的第（n-1）位和第（n+1）位的值是否相同，若这两位的值相同，则第 n 位为 0，否则为 1。

例如，点阵图形第一行的存储字内容为：#35B4，按上述规则生成的第二行的存储字内容为#71B2，其余类推。

0011010110110100 #35B4
0111000110110010 #71B2

[问题]填充下面程序中的{ } ,完成程序，不必要的标号不要写。

[程序]

```
标号    指令码        操作数
        START
        LEA           GR1，0
        LD            GR0，PTN，
                      GR1
LOOP    SLL           GR0，2
                { ① }
                { ② }
                { ③ }
        LEA             { ④ }
                { ⑤ }
                      GR2，-15，
        LEA           GR1
        JMI           LOOP
        EXIT
PTN     DC            #35B4
        DS            15
        END
```

[分析]该程序为字位运算类程序，根据已知字（PTN 数据区首单元的值），求其它 15 个存储单元的值。该程序的结构为一循环结构。共循环 15 次，GR1 作为循环计数器，兼作 PTN 数据区指针，初值为 0，指向 PTN 数据区第一个已知的存储字。程序中①、②、③三个空根据当前存储字求下一存储字的值。④、⑤两空将所有的存储字的值送入 PTN 数据区，并调整 PTN 数据区指针 GR1。程序中各语句的具体分析说明如下：

标号	指令码	操作数	说 明
	START		
	LEA	GR1，0	0→GR1，循环计数器,PTN 数据区
	LD	GR0，PTN，GR1	指针赋初值 已知存储字的值送 GR0
LOOP	SLL	GR0，2	GR0 左移位
		{ ① }	
		{ ② }	根据 GR0 的值求新的存储字
		{ ③ }	
	LEA	{ ④ }	调整 PTN 数据区指针
		{ ⑤ }	所求新的存储字的值送 PTN 数据区
	LEA	GR2，-15，GR1	未循环 15 次转 LOOP
	JMI	LOOP	
	EXIT		
PTN	DC	#35B4	
	DS	15	
	END		

[参考答案]

① EOR GR0，PTN，GR1

② SRL GR0，2

③ SLL GR0，1 或 SLA GR0，1

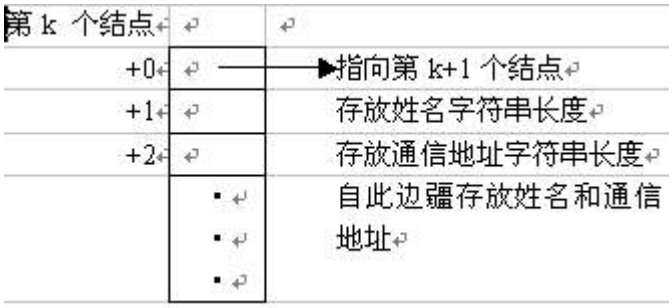
④ GR1，1， GR1

⑤ ST GR0 ，PTN，GR1

[例 7.23]根据姓名查通信地址（链表查询）。

本程序根据输入的姓名(字符串)，在已有的线性表中查找其相应的通信地址(字符串)，并予以输出。

标号为 DATPTR 的存储字存放着线性表第一个结点的地址。结点的数据结构定义为：



最后一个结点的指针字段为空指针（内容为 0）。

[问题]填充程序中的{ }，完成程序，不必要的标号不要写。

[程序]

```
标号    指令码          操作数
        START           BEGIN
OTLONG  DS              1
OTTEXT  DS              80
MOLONG  DC              10
NOTEXT  DC              'NOT FOUND'
INTEXT  DS              80
INLONG  DS              1
DATPTR  DC              FIRST
BEGIN   IN              INTEXT,INLONG
        LEA             GR0,0
        CPA             GR0,INLONG
        JZE             ENDSCH
        LEA             GR3,DATPTR
NXTMEM              { ① }
                    { ② }
        JZE             NOTFND
        LD              GR0,1,GR3
        CPA             GR0,INLONG
        JNZ             NXTMEM
                    { ③ }
        LEA             GR2,INTEXT
        CALL            CMPSTR
                    { ④ }
        LD              GR0,2,GR3
        ST              GR0,OTLONG
```

```
                    { ⑤ }
                    { ⑥ }
        LEA         GR2,OTTEXT
        CALL        CPYSTR
        OUT         OTTEXT,OTLONG
        JMP         BEGIN
NOTFND  OUT         NOTEXT,NOLONG
        JMP         BEGIN
ENDSCH  EXIT
CMPSTR  PUSH        0,GR3
CMPNXT  LD          GR3,0,GR1
        CPL         GR3,0,GR2
        JNZ         CMPEND
        LEA         GR1,1,GR1
        LEA         GR2,1,GR2
        SUB         GR0,CONST1
        JNZ         CMPNXT
CMPEND  POP         GR3
        RET
COMST1  DC          1
CPYSTR  PUSH        0,GR3
CPYNXT  LD          GR3,0,GR1
        ST          GR3,0,GR2
        LEA         GR1,1,GR1
        LEA         GR2,1,GR2
        SUB         GR0,ONE
        JNZ         CPYNXT
        POP         GR3
        RET
ONE     DC          1
        END
```

[分析]这是一个链表查找程序。先输入姓名，然后查找，并输出该姓名对应的地址信息。通用寄存器 GR3 作为指针搜索链表。对程序中指令的分析说明如下：

标 号	指 令 码	操 作 数	说 明
	START	BEGIN	
OTLONG	DS	1	
OTTEXT	DS	80	
MOLONG	DC	10	
NOTEXT	DC	'NOT FOUND'	
INTEXT	DS	80	

INLONG	DS	1	
DATPTR	DC	FIRST	
BEGIN	IN	INTEXT,INLONG	输入姓名信息
	LEA	GR0,0	长度为零转 ENDSCH
	CPA	GR0,INLONG	
	JZE	ENDSCH	
	LEA	GR3,DATPTR	链头指针地址送 GR3
NXTMEM	{ ① }		取链头指针,并判断是否为空指针
	{ ② }		
	JZE	NOTFND	为空指针时转 NOTFND
	LD	GR0,1,GR3	比较输入的姓名长度和所指结点的姓名长度
	CPA	GR0,INLONG	
	JNZ	NXTMEM	不等,转 NXTMEM
	{ ③ }		GR1 指向结点的姓名信息
	LEA	GR2,INTEXT	GR2 指向输入的姓名信息
	CALL	CMPSTR	调 CMPSTR,比较结点的姓名信息和输入的姓名
	{ ④ }		不等,转 NXTMEM
	LD	GR0,2,GR3	通信地址长度送 OTLONG 单元
	ST	GR0,OTLONG	
	{ ⑤ }		GR1 指向通信地址
	{ ⑥ }		
	LEA	GR2,OTTEXT	将通信地址传至 OTTEXT 数据区
	CALL	CPYSTR	
	OUT	OTTEXT,OTLONG	输出通信地址,转 BEGIN
	JMP	BEGIN	
NOTFND	OUT	NOTEXT,NOLONG	输出'NOT FOUND.'信息,转 BEGIN
	JMP	BEGIN	
ENDSCH	EXIT		
CMPSTR	PUSH	0,GR3	(GR3)压栈
CMPNXT	LD	GR3,0,GR1	结点的姓名字符和输入的姓名字符比较
	CPL	GR3,0,GR2	
	JNZ	CMPEND	不等,转 CMPEND
	LEA	GR1,1,GR1	调整指针
	LEA	GR2,1,GR2	
	SUB	GR0,CONST1	是否所有的名字都比较过?否则转 CMPNXT
	JNZ	CMPNXT	
CMPEND	POP	GR3	恢复 GR3 的值
	RET		
COMST1	DC	1	

CPYSTR	PUSH	0,GR3	
CPYNXT	LD	GR3,0,GR1	传达通信地址字符
	ST	GR3,0,GR2	
	LEA	GR1,1,GR1	调整指针
	LEA	GR2,1,GR2	
	SUB	GR0,ONE	是否所有的字符都传送完?否则转 CPYNXT
	JNZ	CPYNXT	
	POP	GR3	
	RET		
ONE	DC	1	
	END		

[答案简析]

① 空可填指令：LD GR3，0，GR3

②空可填指令：LEA GR3， 0，GR3 或 LEA GR0，0，GR3 或 LEA GR1，0，GR3 或 LEA GR2，GR3 或 SLL GR3，0，或 SLA GR3，0 或 SRL GR3，0 或 SRA GR3，0

此外，①、②两空还可以填指令：

LD GR1，0，GR3
LEA GR3， 0，GR1

或 LD GR2， 0，GR3
LEA GR3， 0，GR2

③空可填指令：LEA GR1，3，GR3
在进行姓名信息比较前，使 GR1 指向该结点的姓名信息。

④空可填指令：JNZ NXTMEM
输入的姓名与结点的姓名不同时，则继续查下一结点。

⑤空可填指令：LEA GR1，3，GR3

⑥空可填指令：ADD GR1，1，GR3 或 ADD GR1，INLONG
⑥空可填指令 LEA GR1，INLONG，GR1 就错了，读者可以思考一下为什么。

⑤、⑥两空的指令使 GR1 指向要输出的通信地址。

另外，⑤、⑥两空还可填指令：

```
ADD GR3 1，GR3
LEA GR1，3 GR3

或 ADD GR3 INLONG
LEA GR1，3 GR3
```

7.4.2 应试技巧

水平考试中的汇编试题一般都采用完形填空的形式，即每个汇编试题的程序中留出 8 个左右的空，让考生填上正确的指令，完成程序，这类题目，其难度要小于要求考生独立编程的题目。因为这类题目在程序中已确定了算法，数据结构以及变量。考生在答题时，只要根据程序说明，弄清程序的功能，结合不完整的程序弄清程序中所使用的算法和数据结构，就能够正确填空。但常常有些考生觉得无从下手，不知如何填程序中的空。解答这种类型的题目是否有什么规律或技巧可言呢？我们认为最基本的规律是真正掌握 CASL 及其程序设计。其次了解一点汇编的应试技巧对应试答题帮助极大。

1、 参加应试的考生在掌握 CASL 方面的必须做到以下几点：

（1） 掌握伪指令、宏指令、符号指令的使用方式，理解各条指令的执行功能。

（2） 了解运行 CASL 汇编程序的 COMET 机的硬件结构和 CASL 程序的运行环境。

（3） 掌握标号、字符常数、地址常数、数字常数的构成和使用方式。

（4） 区分取数指令（LD）和取地址指令（LEA）的差别。掌握取地址指令（LEA）的使用以及该指令在循环结构程序中的应用。

（5） 掌握基本数据结构（如，数组，堆栈，表格，链表和二叉树）的构造及其应用。

（6） 掌握一些基本的算法（如，排序，分类和查找等）。

（7） 能进行分支程序，循环程序，主子程序，字符串处理程序和输入/输出等程序的设计与构造。

（8） 能利用堆栈进行递归程序的设计与构造。

我们对 1994、1995 年度的水平考试中的汇编试题作了分析和归纳。从表 7.3 可以看出试题的程序结构和数据结构，从表 7.4 可以看出留空的指令类型。常考的程序结构是分支、循环、输入/输出，字符串处理和字位运算等，常用的数据结构有数组和堆栈。从表 7.4 看出，考得最多的指令类型为取地址指令 LEA。

表 7.3 1994、1995 年度汇编试题结构特点

试	94 程	94 程	94 程	94 高	95 程	95 程	95 程序	95 高级

题	序员级下午试题二计算正整数 N,M 的乘积	序员级下午试题七数字串转换成数值	序员级下午试题十二复原被压缩的数据	级程序员级下午试题六求真分数 a/b 的 N 位小数	序员级下午试题七字符串比较	序员级下午试题二数组对应元素相加	员级下午试题十二输出十六进制数乘法表	程序员级下午试题六自动生成 16×16 点阵图形
程序结构	循 环	循 环	循环	循 环	循环	循 环	双重循环	循 环
数据结构	/	数组	数组	/	数组	数组	数组	/

表 7.4 1994、1995 年度汇编试题留空的指令类型

指令类型	存取数据指令	取地址指令	加减，逻辑运算指令	比较指令	移位指令	转移指令	堆栈指令	操作数
空的个数	9	12	5	1	6	5	1	7

2、 在应试中的几点技巧

（1）注意分析留空前后的指令，因为前后的指令和留空指令关系最紧密。如，在留空指令后出现了一个新的寄存器，而在留空指令前面的指令中未出现过该寄存器，则留空指令和该寄存器有关。

（2）比较指令后的空，一般情条件转移指令。因为比较指令的执行结果只设置 FR，对其它寄存器和内存单元不产生影响。而只有条件转移指令受 FR 影响，其它指令不受 FR 影响。只影响 FR 的指令还有：LEA XR，0，XR 和移动 0 位的左移或右移指令，如，SLL GR，0，SLA，GR，0 等。

（3）标号是理解程序结构的一个标志。如标号 LOOP 是循环结构的标志。表示从 LOOP 开始是一个循环结构，则 LOOP 前的指令必须包含设置初值部分。如，一子程序的入口标号为 DIV，则可猜出该子程序可能是除法子程序。阅读程序时根据标号给出的零星英语含义可帮助理解程序。

总之，应试能力和技巧是一个综合性的问题，只有多多练习，才能逐步摸索出一套适合自身特点的应试程序。

[例 7.24]复原被压缩过的数据。

原始数据的压缩规则如下：

（1） 原始数据中连续出现 N（N≥2）个相同的数 X，则在压缩数据中相继存放 N 和 X 两个数。

（2）在原始数据中，对于不属于上述情况的数据段，即相邻而又不相等的 M 个数据，则在压缩数据中先存放-M，再存放这 M 个数。

（3） 压缩的数据中，数据的存放次序与原始数据中数的出现次序相一致，并在压缩数据的最后加 0，作为结束标志。

例如；原始数据中有以下数：
-15 11 11 11 11 11 11 11 11 -2 14 9 9 9 9 9
则压缩后的数据为：-1 -15 8 11 -2 -2 14 5 9 0

程序中标号为 PAKED 开始的存储字中存放了上述压缩后的数据，程序执行后，在标号为 UNPACK 开始的 16 个存储字中生成原始数据。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

标 号	指 令 码	操 作 数
	START	BEGIN
BEGIN	{ ① }	
	LEA	GR2,0
LOOP0	{ ② }	
	LD	GR3,PACKED,GR1
	LEA	GR3,0,GR3
	{ ③ }	
	{ ④ }	
	LEA	GR1,1,GR1
	LD	GR0,PACKED,GR1
LOOP1	{ ⑤ }	
	LEA	GR2,1,GR2
	LEA	GR3,-1,GR3
	JNZ	LOOP1
	JMP	LOOP0

ONEONE	{ ⑥ }	
LOOP2	LEA	GR1,1,GR1
	LD	GR0,PACKED,GR1
	{ ⑦ }	
	LEA	GR2,1,GR2
	LEA	GR3,-1,GR3
	{ ⑧ }	
	JMP	LOOP0
ENDPGM	EXIT	
UNPACK	DS	16
HXFFFF	DC	#FFFF
PACKED	DC	-1
	DC	-15
	DC	8
	DC	11
	DC	-2
	DC	-2
	DC	14
	DC	5
	DC	9
	DC	0
	END	

[分析]阅读程序，从标号给出的信息可知，该程序由三个循环结构组成：LOOP0，LOOP1，LOOP2。其中 LOOP1 和 LOOP2 这两个循环是并列的。而 LOOP0 循环包含了 LOOP1 和 LOOP2 循环。由标号 ONEONE 再结合程序可以猜出从标号 ONEONE 开始的 LOOP2 循环部分是进行连续 M 个互不相同的数的恢复，对应地 LOOP1 循环是进行连续 N 个相同的数的恢复，LOOP0 循环是进行总控处理，判断是进行 M 个互不相同的数的恢复、还是进行 N 个相同的数的恢复或结束。由此可行出该程处理框图如图 7.33 所示。

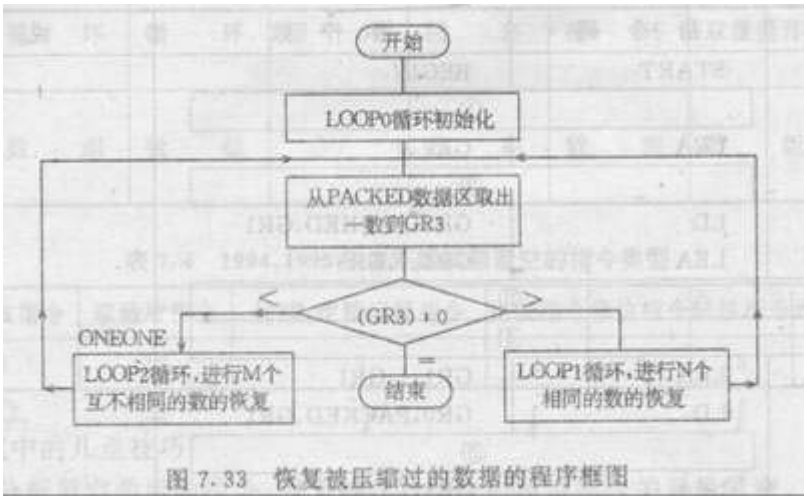


图 7.33 恢复被压缩过的数据的程序框图

结合程序和图 7.33 可知,GR1 为 PACKED 数据区指针;GR2 为 UNPACK 数据区指针;GR3 存放是进行连续 M 个互不相同的数的恢复，还是进行 N 个相同的数的恢复或结束的标志。根据以上的分析可以将图 7.33 进一步细化。由于篇幅限，这里从略。下面进行填空。

[答案简析]

填①、②两空时发现②空下面的一条指令：LD GR3 PACKED ，GR1 使用 GR1 作为 PACKED 数据区的指针。因此①、②两空和 GR1 有关，①空设置 GR1 初值，②空调整 GR1 的值，①空应填指令;
LEA GR1，-1 或 LD GR1，HXFFFF 或 LEA GR1，65535

②空应填指令：
LEA GR1，1，GR1，或 SUB GR1，HXFFFF

③、④两空的上一条指令为 LEA GR3，0，GR3，只设置 FR 的值，判断 GR3 的值为零，为负还是为正。因此，③、④为条件转移指令。当（GR3）为零时，程序结束，当（GR3）为负时，转 ONEONE 进行 M 个互不相同的数的复原。③、④空分别填指令：

JMI ONEONE
JZE ENDPGM

或 JZE ENDPGM
JMI ONEONE

⑤空的上一条指令为 LD GR0，PACKED ，GR1 从 PACKED 数据区取出连续相同的数 X 到 GR0，则该空应将 GR0 的值送 UNPACK 数据区。⑤空应填指令：
ST GR0，UNPACK ，GR2

⑥空的难度较大，由⑧空的上一条指令：LEA GR3，-1 GR3 可知，GR3 为 LOOP2 循环的控制变量，在 LOOP2 循环中 GR3 的值为非负，故在进入 LOOP2 循环前要对

GR3 的值进行调整，将负数变为非负数。⑥空应填指令：
EOR GR3 ，HXFFFF

⑦空的上一条指令为：LD GR0，PACKDE ，GR1 从 PACKED 数据区取出连续互不相同数到 GR0，则该空应填将 GR0 的值传到 UNPACK 数据区的指令：
ST GR0，UNPACK，GR2

⑧空上一条指令 LEA GR3，-1 GR3 调整 LOOP2 循环控制变量 GR3 的值，LOOP2 循环中 GR3 的值非负，故该空应填指令：JPZ LOOP2

习 题

7.1 下面一子程序将大写字母转换成小写小母，该子程序将 GR0 中的大写字母转换成小写字母，如果 GR0 中不是大写字母则原样返回。调用子程序时字符以字符数据的形式存于 GR0 的低八位，GR0 的高八位全为 0。

[问题]填充下面程序中的{ } ，不必要的标号不要写。

[程序]

```
标 号    指 令 码  操 作 数
LOWER    START
         CPA       GR0，CA
               { ① }
         CPA       GR0，AFZ
         JPZ       RT
               { ② }
         RET
         DC          'A'
         DC          '['
         DC        32
         END
```

7．2 取一个字中的某几位。

下面一子程序取出 X 第 P 位开始的有 n 位，并存于 GR0 返回。其中，0≤p≤15、1≤n≤16、p+n≤16。

如 x=#5A7F,p=4,n=4，则该子程序执行后 GR0 中返回#000A。

主程序在 GR1 中给出子程序所需参数 x、p、n 存储区的首地址，如下所示。子程序中寄存器的值不必保护。

(GR0)+0	x
+1	p
+2	n

[问题]填充下程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号      指 令 码        操 作 数
GETBIT     START
           LD              GR2，2，GR1
                    { ① }
           SRA             GR0,-1,GR2
           LD              GR2，1，GR1
                    { ② }
                    { ③ }
           LEA             GR2，16
                    { ④ }
           SUB             GR2，2，GR1
           SRT             GR0，0，GR2
           RET
           END
```

7.3 将输入的信息反序输出。

下面一子程序从终端输入设备上，输入一串信息，然后以反序形式在输出设备上输出。子程序中寄存器的值不必保护。

[问题]填充下面程序中的{ }，完成程序，不必要的标号不要写。

[程序]

```
标 号          指 令 码          操 作 数
REVERS         START
               IN                BUF，LEG
                          { ① }
               LD                GR3，LEG
                          { ② }
LOOP           ST                GR3，WORK
               CPL               GR2，WORK
                          { ③ }
```

```
                LD              GR0，BUF，GR2
                LD              GR1，1，GR3
                              { ④ }
                ST              GR0，BF，GR3
                LEA             GR2，1，GR2
                LEA             GR3，-1，GR3
                JMP             LOOP
PRINT           OUT             BUF，LEG
                RET
BUF             DS              80
LEG             DS              1
WORK            DS              1
                END
```

7．4 借，还书累计

下面一子程序用于累计图书馆中借书，还书的总册数，查询借出图书的总册数；查询库存图书的总册数。累计借书，还书的册数时，由主程序把借书处（有 1，2，3，4 四个借书处）借出或还入的册数放在 GR2 传送给此子程序。查询时，把结果放入 GR2 返回主程序。在处理时 GR1 寄存器的内容和作用为：

字 段	内 容	处 理 类 别	备 注
第 0-7 位	全 0	/	
第 8-11 位	0001	累计借出册数	
	0010	累计还入册数	
	0011	查询借出册数	
	0100	查询库存册数	
第 12-15 位	0001	1 号借书处	累计借出和还入册数时间
	0010	2 号借书处	
	0011	3 号借书处	
	0100	4 号借书处	

在下面的子程序中，LIB 单元用于存放库存册数；RETURN 开始的连续 5 个单元用于存放还书的总册数；LEND 开始的 5 个连续单元用于存放借出的总册数。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号     指 令 码              操 作 数
PRG       START
          ST                    GR1, SAV
          ST                    GR2, NUM
          SRL                   GR1, 4
                          { ① }
          JPZ                   QERY
          LD                    GR3, SAV
                          { ② }
          LD                    GR1, SAV
          SRL                   GR1, 4
          LEA                   GR1, -1, GR1
          JNZ                   L1
          ADD                   GR2, LEND, GR3
                          { ③ }
          LD                    GR2, NUM
          ADD                   GR2, LEND
          ST                    GR2, LEND
          LD                    GR2, LIB
          SUB                   GR2, NUM
          ST                    GR2, LIB
          JMP                   RETN
          ADD                   GR2, RETURN, GR3
          ST                    GR2, RETURN, GR3
          LD                    GR2, NUM
          ADD                   GR2, RETURN
          ST                    GR2, RETURN
          LDD                   GR2, LIB
          ADD                   GR2, NUM
          ST                    GR2, LIB
          JMP                   RETN
QEPY      JNZ                   L2
          LD                    GR2, LEND
          JMP                   RETN
                          { ④ }
RETN      RET
SAV       DS                    1
NUM       DS                    1
LIB       DS                    1
LEND      DS                    5
RETURN    DS                    5
CF        DC                    #000F
```

```
            END
```

7.5 求表达式值。

下面一子程序实现求表达式 $a*10^3+b*10^2+c*10+d$ 的值，a、b、c、d 是 0～9 的数，分别存于 DATA 开始的存储单元。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号       指 令 码              操 作 数
EXP         START
            PUSH                  0，GR1
            LEA                   GR1，0
            LEA                   GR0，0
LOOP        ST                    GR0，WRK
                          { ① }
            ADD                   GR0，WRK
            SLA                   GR0，1
                          { ② }
            LEA                   GR1，1，GR1
            CPA                   GR1，C4
            JMI                   LOOP
                          { ③ }
            RET
DATA        DS                    4
WRK         DS                    1
C4          DC                    4
            END
```

7.6 八进制数输出

下面一子程序将 GR0 的值以八进制形式在终端设备上显示出来。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号       指 令 码              操 作 数
SB          START
```

```
        PUSH                0，GR1，
        ST                  GR0，WRK
        LEA                 GR1，5
LOOP    AND                 GR0，C7
        ADD                 GR0，C30H
                        { ① }
        LEA                 GR1，-1，GR1
        JMI                 L1
                        { ② }
                        { ③ }
        ST                  GR0，WRK
        JMP                 LOOP
L1      OUT                 BUF，LNG
        POP                 GR1
        RET
BUF     DS                  6
WRK     DS                  1
LNG     DC                  6
C7      DC                  #0007
C30H    DC                  #0030
        END
```

7.7 链表反序。

下面一子程序将链表中各结点的排列顺序颠倒过来。调用子程序时 GR1 中存放的是链表头结束点的地址，子程序返回时，GR1 中存放的是反序链表的头结点的地址。链表结点由两个单元组成，链指针占一个单元，链表未结点的指针值为零。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号           指 令 码            操 作 数
REVERS          START
                ST                  GR0，SAVE
                PUSH                0，GR2
                LEA                 GR3，0
LOOP                            { ① }
                LEA                 GR1，0，GR1
                JZE                 NEXT
                                { ② }
                LEA                 GR3，0，GR1
                                { ③ }
```

```
                JMP             LOOP
NEXT            LEA             GR1，0，GR3
                LD              GR0，SAVE
                POP             GR2
                RET
SAVE            DS              1
                END
```

7.8 二项式系数

下面一子程序对于给定的 n、r(2≤n≤15,0≤r≤n)，求二项式系数 $C^r_{n0}C^r_n$ 的计算公式为：

$C^r_n=C^{r-1}_{n-1}+C^r_{n-1}$ （0≤r≤n-1）

$C^0_n=C^r_r=1$

调用该子程序时，参数 r,n 分别存放在 GR1 和 GR2 中，计算结果存于 GR3。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号     指 令 码        操 作 数
CNR       START
          LEA             GR1,0,GR1
          LZE             BACK
          ST              GR2,WRK
                    { ① }
          JZE             BACK
          PUSH            0,GR1
          PUSH            0,GR2
          LEA             GR1,-1,GR1
          LEA             GR2,-1,GR2
                    { ② }
          POP             GR2
          POP             GR1
          PUSH            0,GR3
                    { ③ }
          CALL            CNR
          POP             GR0
          ST              GR0,WRK
                    { ④ }
```

```
        JMP         RTN
BACK    LEA         GR3,1
RTN     RET
WRK     DS          1
        END
```

7.9 最大公约数。

下面一子程序利用下面的四点性质求正整数 M、N 的最大公约数 GCD（M、N）。

（1） 当 M、N 都是偶数时：
GCD（M、N）=2*GCD（M/2，N/2）

（2） 当 M、N 中仅有一个是偶数时，如 M 为偶数时；
GCD（M、N）=GCD（M/2，N）

（3） 当 M、N 都是奇数，且 M>N 时：
GCD（M，N）=GCD（M-N，N）

（4） 当 M=N 时；
GCD（M，N）=M=N

调用子程序时，M、N 的值分别存放在 GR1 和 GR2 中，求得的最大公约数存放在 GR3 返回，子程序中可使用 GR0。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号       指 令 码  操 作 数
GCD         START
            ST        GR2,WRK
            CPA       GR1,WRK
            JZE       LAST
            LEA       GR0,0,GR1
            AND       GR0,C1
              { ① }
            LEA       GR0,0,GR2
            AND       GR0,C1
            JZE       L3
            CPA       GR1,WRK
            JPZ       L1
              { ② }
```

```
            SUB     GR2,WRK
            JMP     L2
L1          SUB     GR1,WRK
L2          CALL    GCD
            JMP     RTN
L3          SRA     GR2,1
            JMP     L5
L4          SRA     GR1,1
            LEA     GR0,0,GR2
            AND     GR0,C1
            JNZ     L5
            SRA     GR2,1
            CALL    GCD
              { ③ }
            JMP     RTN
L5          CALL    GCD
RTN         RET
LAST        LEA     GR3,0,GR1
            JMP     RTN
C1          DC      1
WRK         DS      1
            END
```

7.10 奇偶校验。

下面一子程序用于求出给定的 15 位数据的奇偶校验位。所求的奇偶位是偶校验位，即当给定的 15 位数据中的 1 的个数是奇数时，校验位为 1，否则校验位为 0。主程序把数据放在 GR1 的第 1-15 位，传送给此子程序。求出的校验位置入 GR1 的第 0 位，返回主程序。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号     指 令 码     操 作 数
SB        START
          PUSH        0,GR2
          PUSH        0,GR3
          ST          GR1,SAVE
          LEA         GR2,0
          LEA         GR3,14
LOOP      ST          GR1,WRK
          AND         GR1,C1
```

```
                { ① }
        ST      GR1,C0
                { ② }
        SRL     GR1,1
        LEA     GR3,-1,GR3
        JPZ     LOOP
        LD      GR1,SAVE
        LD      GR2,C0
                { ③ }
        JZE     L1
        OR      GR1,C8000
        JMP     L2
L1      AND     GR1,C7FFF
L2      POP     GR3
        POP     GR2
        RET
SAVE    DS      1
WRK     DS      1
C0      DC      0
C1      DC      1
C8000   DC      #8000
C7FFF   DC      #7FFF
        END
```

7.11 十六进制数输出

下面一子程序将 GR1 中的值转换成四位十六进制数在输出设备上输出。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号   指 令 码   操 作 数
SB      START
        LEA     GR2,4
        ST      GR2,BLENG
                { ① }
        JMI     LAST
        LEA     GR0,0,GR1
                { ② }
        ADD     GR0,C0
        CPA     GR0,C9
```

```
        JMI     L1
        ADD     GR0,C07
L1      ST      GR0,BUFFET,GR2
                { ③ }
        JMP     LOOP
                { ④ }
        RET
BUFFER  DS      80
BLENG   DS      1
CF      DC      #000F
C9      DC      ':'
C07     DC      7
C0      DC      '0'
        END
```

7.12 商品检验码。

下面是一个求商品码的检查数字的程序。商品码由 13 位数字组成，第 13 位检查数字，它由第 1-12 位数字进行还原计算得到。

例如：4 9 0 1 2 3 4 5 6 7 8 9 4←检查数字

检查数字按下面的方法计算

（1）1-12 位的各偶数位求和再乘 3。如
（9+1+1+3+5+7+9）×3=102

（2）1-12 位的各奇数求和。如：
4+0+2+4+6+8=24

（3）上面的（1）和（2）的值相加。如：
102+24=126

（4）上面（3）得到的值除以 10 求其余数。如：
126÷10=12 余 6。

（5） 从 10 中减去（4）得到的余数。如 10-6=4。这里得到的数就是检查数字。但当其结果为 10 时，当作 0。

（6） 下面的程序是输入除检查数字以外的 12 位商品码，计算检查数字并予以显示的程序。不输入商品码而输入回车符号时，本程序终了。若输入的商品码是数字以外的符号，或者输入位数不是 12 位，那就给出"INPUT ERROR"信息，要求重新输入。

[问题]填充下面程序中的{ } ，完成程序，不必要的标号不要写。

[程序]

```
标 号  指 令 码     操 作 数
P62SD  START
BEGIN  IN           INBUF,LENG
       LEA          GR3,0
       CPA          GR3,LENG

       JZE          LAST

       LEA          GR3,12
       CPA          GR3,LENG
       JZE          LABEL1
ERROR  OUT          ERRMSG,C10
       JMP          BEGIN
LABEL1 LEA          GR1,0
LOOP1  LD           GR2,INBUF,GR1
       CPL          GR2,ZERO
                  { ① }
       CPL          GR2,TEN
       JPZ          ERROR
LABEL2 AND          GR2,MASK1
       ST           GR2,DATA,GR1
       CPA          GR2,C11
       JZE          LABEL3
                  { ② }
       JMP          LOOP1
LABEL3 LEA          GR2,0
       LEA          GR1,1
LOOP2  ADD          GR2,DATA,GR1
       CPA          GR1,C11
       JZE          LABEL4
       LEA          GR1,2,GR1
       JMP          LOOP2
                  { ③ }
       ADD          GR2,WORK
       ADD          GR2,WORK
       LEA          GR1,0
LOOP3  ADD          GR2,DATA,GR1
       CPA          GR1,C10
       JZE          LABEL5
       LEA          GR1,2,GR1
       JMP          LOOP3
```

```
LABEL5 LEA      GR2,10,GR2
LABEL6 CPA      GR2,C11
                { ④ }
       LEA      GR2,-10,GR2
       JMP      LABEL6
LABEL7 ST       GR2,WORK
       LEA      GR3,10
       SUB      GR3,WORK
                { ⑤ }
       ST       GR3,CHKDGT
       OUT      RESULT,C15
       JMP      BEGIN
LAST   EXIT
INBUF  DS       80
LENG   DS       1
ZERO   DC       '0'
TEN    DC       ':'
ERRMSG DC       'INPUT ERROR'
C10    DC       10
MASK1  DC       #000F
MASK2  DC       #0030
DATA   DS       12
WORK   DS       1
C11    DC       11
RESULT DC       'THE RESULT
                IS'
CHKDGT DS       1
C15    DC       15
       END
```

CASL 汇编语言程序设计(教材)

第 1 章 数制及运算系统

本章所论及的内容是汇编语言程序设计所需要的必备知识。如果读者业已熟悉它们,可以跳过而进入第 2 章的学习。

无论计算机从外部世界接受的是什么样的数据，例如十进制数，字符常数，甚至图示，它们在计算机的内部形式都是二进制代码.计算机采用二进制数来表达数据是因为具有两个稳定的物理状态，在电子线路上容易实现的缘故。计算机中的

数据也可以是指令的编码,指令经过译码，产生一列操作，实现各种运算。计算机只有通过指令的执行，把内部形式的代码解释成特定的含义，才能将各种代码当作有意义的信息来使用。

计算机要进行数据的加工,必然涉及到数据在不同进位计数制下的表示方法和它们之间的转换，不同编码方式的数据在计算机上的形态，以及它与通常意义下数的表示之间的关系。要进行数据的运算，还涉及到采用的何种运算系统。

1.1 进位计数制

进位计数制用于计算机及人们日常事务中。同一含义的数，在不同进位制下有不同的表示方法。虽然都是由一串数码组成，但数制基不同，就有不同的解释。例如，数码串 101，在二进制下它的值是 5，但在十六进制下它的值是 257，因此有必要研究进位计数制及其性质.

假定整数 N，它在一般进位计数制下，每一个数位都对应一个权 R^i，这里的 R 称为数制的基数或底。因此,N 在这样数制系统中,它的一般形式为:

$$N=a_{p-1}a_{p-2}\cdots a_1a_0$$

a_i (i=0,1, …p-1)是下列多项式的系数

$$N=a_{p-1}\times R^{p-1}+a_{p-2}\times R^{p-2}+\cdots+a_1\times R^1+a_0\times R^0$$

$$=\sum_{i=0}^{p-1}a_iR^i$$

在 R 进位计数制系统中，每个 R 进位制数具有以下性质：

（1）有 R 个数码。即 a_i 可为 0，1，…，R—2，R—1 中任何一个数码，组成 R 进制数的数值部分.

（2）逢 R 进 1。任意位置上的 R 个单位，构成高一位置上的 1 个单位。同样，任一位置上的 1 个单位，是它低一位置（右边）上的 1 个单位的 R 倍。

（3）数值唯一。当基数 R 确定时,则一个 R 进制表示的数，在十进制记数系统中的数值是唯一确定的.因为:

$$N=\sum_{i=0}^{p-1}a_iR^i$$

其中,$0<a_i<R-1$ (i=0,1,…p-1)， 且 $a_{p-1}\neq 0$。

人们习惯使用十进制数，而几乎所有计算机都是用二进制，势必要了解数的二进制表示及其性质，进而懂得二进制算术运算的规则和特点。由于二进制书写很不方便，人们又引进八进制或十六进位计数制。以八进制或十六进制形式来表示数，是因为它们的基数恰好是 2 的乘方，特别适合于表达二进制数。这样在十进制数和二进制数之间的转换，藉助于八进制或十六进制表示数是很自然的了。大多数计算机软件不是用八进制就是用十六进制描述机器中的二进制数。例如 PDP－11 是用八进制，而在本书 COMET 计算机用的是十六进制。选用八进制还是十六进制表示完全是一种文件的习惯。选用什么对硬件的影响，仅在于计算机操作面板上的灯和开关是排成三个一组还是四个一组。

在懂得一般进位计数制的性质之后，对于特定进位制，只需对基数 R 给出确定的值（十进制记数系统意义下），就类似地给出它的表示和它的性质。不妨看一下人们熟知的十进制数的表示。

十进制数是以 10 为基，它具有：

(1)有 10 个数码。即用 0，1，2，3，4，5，6，7，8，9 这十个数码表示十进制数的数值部分.

(2)逢 10 进 1. 这是因为十进制系统中以 10 为基,加法的满 10 进 1,减法的借 1 当 10,数值 825 最左边的数字 8 是"百倍",具有 8×100=800 的含义,都是由这个性质决定的.

(3)数值唯一. 一个数 N 在以 10 为基,可用以下多项式表达:

$$N=\sum_{i=0}^{p-1}a_i(10)^i$$

其中系数 a_i(i=0,1,…p-1) 组成

$$N=a_{p-1}a_{p-2}\cdots a_1a_0(\text{假定 } a_{p-1}\neq 0)$$

就是数 N 的十进制表示.按 10 的幂和相应的 a_i 相乘,逐个相加,其和就是数 N 唯一确定的值.例如

$N=8\times10^3+1\times10^2+9\times10^1+2\times10^0$
$=8192$

这说明任何一个数 N 都可分解成一个以 10 为基的多项式,在十进制记数系统中,经过运算,其值 8192 唯一确定.

二进制数是以 2 为基，它具有：

（1）有两个数码，即 0，1 表示二进制数的数值部分。

（2）逢 2 进 1。基于这一特性，人们运用逻辑代数原理，进行计算机的逻辑设计.

（3）任何一个数 N，总可以用 2 为基的多项式表达：

$$N=\sum_{i=0}^{p-1} b_i(2)^i$$

其中系数 b_i（i=0,1,…p-1）组成

$$N=b_{p-1}b_{p-2}\cdots b_1b_0\text{(假定 bp-1}\neq 0\text{)}$$

就是数 N 的十进制表示.同样原理,任何一个二进制数

$$B=b_{k-1}b_{k-2}\cdots b_1b_0$$

展开成以 2 为基的多项式

$$B=\sum_{i=0}^{k-1} b_i(2)^i$$

在十进制记数中,进行以 2 为幂的乘加运算,就可唯一得到它的数值.例如:

B=11001010
$=1\times 2^7+1\times 2^6+0\times 2^5+0\times 2^4+1\times 2^3+0\times 2^2+1\times 2^1+0\times 2^0$
=128+64+8+2
=202

由性质（3），实际上也给出一个十进制数如何表达成二进制数的方法。

1.2 二进制数的运算

根据二进制数性质（2），可以类似于十进制系统建立二进制运算规则。只要记住“逢 2 进 1，借 1 当 2”，就可进行二进制数的算术运算。二进制数的加减手算法是和我们在小学学过的算术一样的，只不过得到的加法和减法表不同而已。表 1．1 是二进制数的加法和减法表。

表 1.1 二进制加法和减法表

Cin,Bin	X	Y	X+Y+Cin	Cout	X-Y-Bin	bout
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	0

0	1	1	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	1	0	0	1	0	0
1	1	1	1	1	1	1

按照此表，两个二进制数 X 和 Y 的相加，实际上是将它们的最小有效位和初始进位以(Cin)零相加，产生(X+Y+Cin)与进位(Cout).接着从右到左运算每一位，计算时要把每一列的进位加到下一列的和中去。十进制加法和相应的二进制加法的例子表示如下，进位用一个计由 C 表示。

进位 C
被加数 X
加数 Y
相加 X+Y

```
  190          101111000
+ 141            10111110
-----        +   10001101
  331          ---------
               101001011
```

进位 C
被加数 X
加数 Y
相加 X+Y

```
  173          001011000
+  44            10101101
-----        +   00101100
  217          ---------
               11011001
```

减法是类似的,但要用借位(bin 和 bout)代替进位:

借位 B
被减数 X
减数 Y
相减 X-Y

```
  229          001111100
-  46            11100101
-----        -   00101110
  183          ---------
               10110111
```

在计算机中，减法通常的用途是比较两个数。例如，若（X—Y）的运算产生一个超出最高有效位的借位,则 X 小于 Y；否则 X 大于或等于 Y。

加法和减法表可扩展到八进制和十六进制数，也可扩展到任何一种基数。然而，由于种种理由，几乎没有计算机科学工作者费心去记忆这些表。减法表完全不必要，因为减法总可以用补码加法来实现，这将在下一节中说明。甚至加法也很少用手算，特别是在高级语言程序设计中。

COMET 计算机的终端上输出格式是采用十六进制数形式来表示机内二进制代码，这在调试程序或需要观察程序在机内的代码,需要做一点十六进制加或减法的运算。以下 仅给出一个十六进制数相加的示例。

两个十六进制数相加的心算步骤如下：

进位 C
被加数 X

```
 110  0          1   1    0    0
 19B  9(16)      1   9    11   9
```

加数　　Y
相加　　X+Y

二进制数的乘法和除法，在许多计算机中是用机器指令来实现的。像有些小型计算机为了节省硬件而省略了乘除指令，COMET 计算机没有设置这类指令，因此必须在软件中用加法或减法指令和移位来实现。

在小学我们就学过乘法，它是由乘数的每一位数分别和被乘数相乘得到，如下所示：

```
   11
 × 13
 ----
   33
  11
 ----
  143

   1011
 × 1101
 ------
   1011
  0000
 1011
1011
--------
10001111
```

在二进制乘法中得到移位的被乘数是很简单的，因为乘数的值只可能是 0 和 1。

不过，在计算机中不是列出所有的移位的被乘数再相加，而是采用将每一次得到的移位的被乘数和部分积相加的办法做乘法.用这种方法再次重复上述例子示于表 1．2 中。

表 1.2 用部分积的方法做的乘法

1011	被乘数
× 1101	乘数
00000000	部分积
1011	移位的被乘数
00001011	部分积
0000	移位的被乘数
00001011	部分积
1011	移位的被乘数
00110111	部分积
1011	移位的被乘数
10001111	乘数

最简单的二进制除法算法也是建立在小学学过的方法的基础上，如表 1．3 所示。在十进制和二进制中，我们都先将余数和除数的倍数进行比较，以便确定从余数中可以减去多少倍移位的除数。在十进制的例子中，我们第一步选 11 作为 21 的最小倍数，然后选 99 作为小于 107 的 11 的最小倍数。在二进制中，选定除数

倍数要简单一些，因为只有两种选择，不是 0 就是除数本身。但是为了选择适当的移位除数，比较操作仍然是需要的。

表 1.3 长除的例子

	10011	商
110011	11011001	被除数
	1011	移位的除数
	0101	减后的被乘数
	0000	移位的除数
	1010	减后的被乘数
	0000	移位的除数
	10100	减后的被乘数
	1011	移位的除数
	10011	减后的被乘数
	1011	移位的除数
	1000	余数

在计算机中，不带符号的除法是和乘法互补的。一个典型的除法算法接收一个双字节的被除数和一个单字节的除数，产生单字节的商和余数。如果除数为 0 或商大于一个字节，则该除法将产生溢出。第二种情况仅发生在除数小于或等于被除数的高位字节时。

以上所讨论的二进制数的加、减、乘、除的规则，是针对无符号的二进制代码而言的。有符号数的加减法，虽然计算机视为无符号的二进制代码进行加减，但要求程序给出适当的解释。有符号数的乘除与无符号数的乘除算法是有差异的，以后再去讨论它们

1·3 数制之间的转换

程序员需要把一个机内数输出，首先是把这些数表示为十进制形式，那将方便得多。有时需要察看机内代码，用十六进制形式输出，不仅读起来方便，而且还与二进制之间保持有良好的对应关系（十进制不存在这种对应），这对了解指令编码或机内特征位是很必要的。同样，把十进制数或十六进制数转换为二进制形式也是需要的。一般来讲，两种基数之间的转换不能用简单的置换法来实现，需要通过一定的运算。

首先给出十进制、二进制、八进制、十六进制的数码之间的对应关系，然后再分别讨论它们数的表达形式之间的转换方法。

1. 二进制→十六进制的转换（记为 2→16）

二进制的基是 2，十六进制的基是 16，恰好是 2 的 4 次乘方，即 4 位二进制数正好对应 1 位十六进制数。这样我们从最低位（右边）开始，把二进制数分成 4 位一组，每组都用一个等效的十六进制数字代替即可。

例如：

二进制数	0110	1010	1101	1111
十六进制数	6	A	D	F

上述过程中应注意到，在二进制数左边（高位）可根据需要随意地加 0（对于无符号数而言），以便使总的位数正好是 4 的倍数。

2．十六进制→二进制的转换（记为 16→2）

和 2→16 过程正好相反以,16→2 是用等效于十六进制数字的 4 位一组二进制数来代替一个十六进制数字，连起来一串二进位数字组成一个二进制数。

例如：

十六进制数	1	2	A	6
二进制数	0001	0010	1010	0110

应该注意,不管表示一个十六进制数字实际上要多少二进位，重要的是每个十六进制数字必须
被 4 位二进位所代替,如数字 2,是用 4 位 0010 替代,而不是仅用 2 位的 10 来替代,否则得不到正确的二进制结果.

3. 十六进制→十进制的转换（记为 16→10）

十六进制数转换成十进制数,从本质上说是将它表达成以 10 为基数的多项式

$N_{(16)}=\sum ai \cdot (10)^{i}$ (i=0,1,…,p-1)

则 $a_{p-1}a_{p-2}\cdots a1a0$(设 $a_{p-1}\neq 0$)即为 $N_{(16)}$的十进制数,但是求出这个多项式的各项系数 ai(i=0,1,…,p-1)不是很方便.在十进制记数系统运算以 16 为基的多项式的值就是 $N_{(16)}$的十进制表示的数.这种方法简便,只要将数的各位按 16 的幂次展开,然后按十进制规则,逐次进行乘法运算即得.

例如：$1BE8_{(16)}=1\times16^{3}+11\times16^{2}+14\times16^{1}\times+8\times16^{0}$
$=4096+2816+224+8$
$=7144_{(10)}$

16→2 另一种方法，可查十六进制和十进制转换表（见表 1．5），形如 h4h3h2h1 的四位十六进制数，分别从表 1．5 中查出各位对应的十进制值，然后做个十进制加法即可得到。例如：十六制数 7FDE，查表如下：

h4=7 对应 28672
h3=F 对应 3840

h2=D 对应 208
h1=E 对应 14

其和为:32734

这种方法对较大的十六进制数,只需查表做简单的加法即可得到对应的十进制数.

表 1.5 十六进制和十进制转换表

4		3		2		1	
十六	=+	十六	=+	十六	=+	十六	=+
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

4. 十进制→十六进制的转换（记为 10→16）

在进位计数值一节中，根据性质（3），取基数 R=16，则任何一个十进制数 N，可唯一表达成

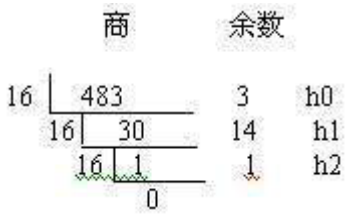
$N=h_m \cdot 16^m+h_{m-1} \cdot 16^{m-1}+\cdots + h1 \cdot 16 + h0$

试想，假定用 16 来除这个公式中的各数，发生了什么情况。由带余除法定理，得到：

商 $Q=h_m \cdot 16^{m-1}+ h_{m-1} \cdot 16^{m-2}+\cdots h2 \cdot 16+h1$

余数 $R0=h0 \quad 0 \leqslant h0 \leqslant 16$

例如，将十进制数 483 转换为十六进制数形式，有以下过程



所以 $483_{(10)}=1E3_{(16)}$

10→16 第二种方法也可以应用表 1.5,在十六进制和十进制转换表中找到小于该十进制数的最大表值，找到对应的十六进制数字所在的列，求差，再去查表，直到低位。

5．二进制→十进制的转换（记为 2→10）

像 16→10 那样，二进制数转换成十进制数的最直接方法，是把二进制数按基数 2 的冥次展开．按十进制记数系统的规则，逐项求和，其结果就是该数的十进制表示。例如

$10101_{(2)}=1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0$
$=16+4+1$
$=21_{(10)}$

6．十进制→二进制的转换（记为 10→2）

按 10→16 的原理，十进制数转换为二进制数方法是进行 2 的多次除法，从低位到高位逐个取余数，直到商为零止，最后的余数就是二进制数的最高位，等等。

2→10 和 10→2 的第二种方法是通过 16→10 与 10→16，即先把二进制数转换成十六进制数，再由十六进制数转换成十进制数，它的道运算也如此。如果任意一个 4 位二进制码和十六进制数字对应熟悉的话，第二种方法不失为一个简便的方法。

7．实用的 10→2

程序中需要的数据，程序员可利用汇编语言的伪指令语句“DC n”，它由汇编程序自动将 n 所代表的十进制数或十六进制数转换成机内二进制数。

如果程序中需要动态地从输入设备上(例如键盘)输入十进制数，又是怎样的呢？

我们注意到上述的数制间的转换过程，所有数据是作为数值本身的转换。实际上程序员从键盘上输入的数据，例如一个十进制数是作为字符形式的数字串而存在的。要把一个十进制数字串 $a_{p-1}a_{p-2}\cdots a1a0$，变成为真正数值来用，要经过如下两步：

第一步：将输入的数字字符换成该数字的面值。因为一个字符的 ASCll 编码由 7 位二进制位组成。十进制数字字符 ASCll 码的低 4 位即为其面值，截下 4 位即为它对应的二进制值。

第二步：从高位到低位，利用下列公式进行“乘加”操作，即得到 $a_{p-1}a_{p-2}\cdots a1a0$ 所代表的二进制值。

$N=(\cdots((a_{p-1})\times10+a_{p-2})\times10+\cdots+a1)\times10+a0$

其中每次相乘的基数 10 应是二进制数 1010 参加运算。

这个公式看来似乎并没有太大的价值，但它却是给出一个实用的 10→2 算法。

1.4 数在计算机上的表示

在计算机内的一切数据都是二进制代码，单纯的数据本身并无实际意义，这些代码只有给出特定解释，才能作为有用的数据使用。现在要问，各式各样的数据以怎样的方式放进计算机中？到目前为止，我们只涉及了正整数，确切地说是无符号整数。无符号整数的二进制形式和机内二进制形式是完全一致的。在日常事务中，我们常常用到带符号的数值系统，这就是讨论负数在计算机中如何表达。负数的表达有许多种方法，但是多数计算机采用的是补码系统，因此本节重点是讨论数的补码表示及补码运算系统的一般特性。

1．4．1 有符号整数的表示

有符号数在计算机中的表示，关键是用什么方式将它的符号放进去。根据二进制数码具有 0、1 两个状态，我们可以约定正号“十”和负号“一”分别用数码 0,1 两个状态来代表，这叫符号的数码化。既然符号要占去一个二进位，那末一个 n＋1 位字长的数值部分最多只能是 n 位了。为了区别原来的数与它在计算机中表示的数，把经过“符号数码化”的数称为机器码，原来的数称为机器码的真值。

1．符号-幅值表示法

用符号一幅值表示的机器码称为原码，其定义是：假定整数 X 是 n 位二进制数码的数,即

$$X=\pm X1X2\cdots Xn$$

$$[X]\text{原码} = \begin{cases} X & \text{当 } 2^n>X\geqslant0 \\ 2^n-X & \text{当 } 0\geqslant X>-2^n \end{cases}$$

按此定义，我们看十进制数十 43 和一 127 的原码是怎样的？不妨假定把它们用一个 8 位二进制位
（一个字节）来表达：

$A=+43_{(10)}=+0101011_{(2)}$

$[A]$原码$=00101011_{(2)}$

$B=-127_{(10)}=-1111111_{(2)}$

$[B]$原码$=11111111_{(2)}$

原码用二进制数表示很容易，只需在数值前面以一个附加符号位置 0 或 1，当数值为正时置 0，数值为负时置 1，既方便又容易读出它的真值，若要这个数变号，只需改变它的符号位。

原码系统具有相同数量的正数和负数。一个 n 位的有符号数一定包括在$-(2^n-1)$到$+(2^n-1)$的范围内，包括 0 的两种可能的表示法。

原码系统有两个缺点：

一是 0 的不唯一性，由定义：

[+0]原码=00…0 (n+1 个 0)

[-0]原码=10…0 (n 个 0)

二是符号位仅作正负标志，不能参加运算。若要做原码的加减，要判别符号位等额外操作。这带来逻辑线路复杂性，是目前计算机不采用原码系统的主要原因。

2．补码表示法

补码定义：假定具有 n 位二进制数码的数 X=±X1X2…Xn

$$[X]\text{补码}=\begin{cases} X & \text{当 } 2^n>X\geqslant 0 \\ 2^{n+1}+X & \text{当 } 0\geqslant X>-2^n \end{cases}$$

记[X]补码=y0y1…yn 是 n＋1 位机器码。不妨设 n=7，看十进制数十 125 和一 109 的 8 位机器码表示的补码是什么？由定义，

令 A=+125=+1111101,A>0

[A]补码=01111101

令 B=-109=-1101101 B<0

[X]补码$=2^8+(-1101101)=10010011$

用补码表示的机器码，正数和它的真值是一致的，但负数的补码和它的真值的数值部分差异甚大，似乎不直观，不方便。但是，补码系统却有许多很好的性质，目前大多数计算机是采用补码系统。

真值的正负属性由机器码的符号位 y0 反映，y0=0，真值为正，y0=1，真值为负。零的补码表示是唯一的，并且确认为正的。因为

[+0]补码=00…0 (n+1 个 0)

[-0]补码=00…0 (n+1 个 0)

注意到-0 的补码在 n＋1 位机器上只能保留 n＋1 个 0，而它左边的 1 是应该丢弃的，这样在 mod 2^{n+1} 意义下，两者是相同的。补码系统中正数和负数的个数不对称，一个 n 位的有符号数，包括在一 2^n 到十（2^n 一 1）的范围内，一个很负的数一 2^n 没有对应的正数。可以说，除了一 2^n 的补码没有相反数外其他都可作变号的运算（取补操作）。

补码定义是在同余概念下建立的，可以说[X]补码和真值 X 对模 2^{n+1} 是同余的，因此补码的定义亦可以表示为：

[X]补码=X (mod 2^{n+1})

例如：X=一 101101 在 8 位字节上的补码是

[X]补码=2^8+(-101101)=11010011

改写一下，[X]补码=0×2^8+11010011,用 2^8 去除，显然它的余数是 11010011 而真值 X 可改写成

X=-101101=-1×2^8+11010011

同样对 X 用 2^8 去除，其余数也是 11010011，所以有

[X]补码=-101101(mod 2^8)=11010011

说明 X 和[X]补码对模 2^8 是同余的，即它们分别用 2^8 去整除，所得的余数是相同的（按整除定义觇定余数一定是大于等于零）。

在同余意义下，整数的补码加减法运算，其符号可像数值一样参加运算，而且运算结果在 mod 2^{n+1} 意义下是相等的。许多计算机采用补码系统运算，主要是这个缘故。

例如上述示例中,A=125，B=一 109

A+B=+125+(-109)=16>0

[A+B]补码=00010000

[A]补码+[B]补码=01111101+10010011

=100010000=00010000(mod 2^8)

下一节我们还可以看到补码运算系统其它的一些特性。

3·反码表示法

反码定义：假定 X=±X1X2…Xn

$$[X]_{反码} = \begin{cases} X & 当\ 2^n > X \geqslant 0 \\ (2^{n+1}-1)+X & 当\ 0 \geqslant X > -2^n \end{cases}$$

n 位数的 n 十 1 位机器码表示的反码，同样以最高位反映真值的正负性，即机器码的最高位为 0，真值为正，最高值为 1，真值为负，例如：

A=+1111101 >0 假定 n=7 时

[A]反码=01111101

B=-1101101 <0

[B]反码=(2^8-1)-1101101

=11111111-1101101

=10010010

正数的补码和反码是一样的，负数的表示两者相差 1。反码所能表示的数值范围是-（2^n-1）到+（2^n-1），正数和负数反码的个数相等。0 也有两种表示，即+0（00…0）和-0（11…1）。

反码系统的优点是它具有对称性，即用反码表示的数变号很容易实现，只要把各个二进位取反，0 变 1，1 变 0 即可。例 X=+119=+$1110111_{(2)}$则[X]反码=01110111，若需把 X 变号，只要把[X]反码的各位取反即得到[-X]反码=10001000，这正好就是十进制数一 119 的反码表示。

但是 0 的不唯一性，给运算带来麻烦。除 CDC 公司的某些计算机中使用外，其它新的 计算机很少采用反码系统。但在某些检查和软件应用中，它仍然是有用的。

4．移码表示法

移码又称增码，它的定义是：假定 X=±X1X2…Xn

[X]移码=2^n+X $-2^n \leqslant X < 2^n$

移码总是在其值(n 位二进位)基础上加上 2^n，在需要使用真值时，总是先要减去一个 2^n，因此移码表示法又称为余 2^n 码表示法。移码表示的 n＋1 位机器码能代

表一个范围从一 2^n 到+（2^n-1）之间的数 X，而且正好把 X 的-2^n 到+（2^n-1）范围映照到 0 到（2^{n+1}-1）的无符号数范围。这个范围和 X 的补码表示的范围一致，但符号位总相反。移码表示法用在表示浮点数制的阶码部分。

我们用表 1．6 给出《位机器码分别表示各种编码的对照情况。可见，同一真值在不同系统中可能有不同的机器值。同一机器码，在不同系统有不同的解释。

表 1.6 4 位的补码、反码、原码、移码表示对照表

十进制	补码	反码	原码	移码
-8	1000	-	-	0000
-7	1001	1000	1111	0001
-6	1010	1001	1110	0010
-5	1011	1010	1101	0011
-4	1100	1011	1100	0100
-3	1101	1100	1011	0101
-2	1110	1101	1010	0110
-1	1111	1110	1001	0111
		1111	1000	
0	0000	或	或	1000
		0000	0000	
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

1.4.2 浮点数的表示

以上讨论的是定点机上的整数表示，它是默认小数点固定在最右边那位的右侧。如果把小数点固定在最高位的右侧，就可以表示小数，例如：

$1.100\cdots\cdots0_{(2)}=1.5_{(10)}$
$0.010\cdots\cdots0_{(2)}=0.25_{(10)}$
$1.11\cdots\cdots0_{(2)}=1.999969482421875_{(10)}$

等等。按照进位计数制原理，小数点右边各位的权是 2 的负幂，同样可以进行运算。不过，固定小数点的位置，进行运算时要程序员记住隐比例因子，一般都不希望采用这方法。在定点机上可以用浮点表示法来对每个数的比例因子进行编码。例如在 16 位字长中，如下图示分成两段，一段表示价码 J，一般表示尾数 W，

并且拿出一位表示数符。在这种格式中
的一个数 X 的值是

X=士 2^{J}. W

其中 W 和 J 采用经过编码的无符号二进制，显比例因子就是 2^{J}。

0	1	5 6	15
f	J	W	

例如 $X=46.5625_{(10)}=2E\cdot 9_{(16)}=\cdot 2E9\times 16^{2}$
$=2^{6}\times 0.1011101001$

在实际应用中，都是以 2 个字长，4 个字长，甚至 8 个字长来表示一个浮点数。显然阶码大了，可表示数值范围更大：尾数长了，可以有更高的精度。为了表示小数，允许阶码为负。不妨以下面两个字长定义的浮点格式来表示可能范围的一切实数。

0 1	8 9	15 0	15
f J	W		W

其中,f 为数符，0 为正,J 为负,W 为尾数，共 23 位，无符号二进制代码 J 为阶码共 8 位，用移码表示法表示土 j（一 128≤j≤＋127）。当 W 满足 0. 5≤W≤1 时称为规格化的浮点表示法。因此非 0 的尾数第 1 位总是 1，0 的规格化是全 0。用规格化表示法的浮点表示是唯一的。为了确保运算的精度，要求参加运算的数总是经过规格化处理的。

用 8 位机器码的移码表示的阶码数值范围应是：

$$-128\leqslant j\leqslant +127$$

用余 128 编码（2^{7}）方法，$J=2^{7}$十 1，正好映照成 J 的范围为：

$00000000_{(2)}\sim 11111111_{(2)}$

在这个意义下，X=46. 5625 的各段是

f=0

J=10000110

W=10111010010…0（共 23 位）

用十六进制书写的浮点数是 435D2000

又：$y=-0.125_{(10)}$的浮点格式有如下表示

因为 $y=-0.125_{(10)}=-0.001_{(2)}=-2^{-2}\times 0.1_{(2)}$,

因此 j=-2，它的移码 $J=2^7+(-2)=01111110$

W=100…0，f=1

组合后,yJ 的浮点表示是 BF40 0000 的十六进制代码。

一个定点计算机如果配有浮点解释系统软件，即可方便地进行浮点的运算.事实上,有许多计算机已有浮点指令的硬件配置，可直接使用浮点指令运算。

1·4·3 字符码的表示

字符也是一种数据,它以特定形式载入计算机中;反之,计算机的某些二进制数也可以解释成相应的字符.把字符用特定的二进制数表示,给人们和计算机交换信息带来很大方便.事实上,大多数当代计算机的输出是非数字的.最常见的非数字数据类型是文本,即由某些字符组成的字符串.在计算机中每个字符是按照已有的惯例由一个位串表示.

最常见的字符码是 ASCll 码（the AmeriCan Standard Code for Information Interex一
change）。ASCll 码用一个 7 位串表示每一个字符，总共有 128 个不同的字符（见附录 A）。
于是，字符串“Yeccch！”是非常简单的七个 7 位数表示的：

1011001 1100101 1100011 1100011 1100011 1101000 0100001

在 ASCll 码中有些 7 位串是表示设备的控制功能而不是“打印＂字符,例如 CR（0001101）是使打印机的印刷头或显示器的光标返回到第一列，而 LF（000010）是使打印头或光标进到下一行。其余的大多数控制字符是用于数据通信连接装置。但不同的计算机系统这些字符有不同的功能。

全部字符不超过 128 个.因此它们的 ASCII 码在一个字节里的有效位最多只要 7 位二进位,就可以区分了.这样,一个字节的最高位就不是字符编码的有效位,但是这一位可被用来作为校验位.通常使用的是奇偶校验法,故称字节的最高位为奇偶校验位.当采用偶校验方法时,使校验和其它有效位的 1 的个数保持偶数.例如:字母 C 的 ASCII 码为 $1000011_{(2)}$,其中 1 的个数是奇数,校验位应为 1,使它在机器存储过程中为 11000011.这样字符代码在传输过程中如果有一位出错,则字节中 1 的个数就不是偶数了.一般情况下,奇偶校验可以用来检查字符传输的错误.校验位不管是 1 还是 0,对字符的 I/O 不发生影响.

1.4.4 二--十进制编码的表示

二--十进制编码又称为 BCD 码（Binary－Coded Decimal）。在实际使用中，读出二进制数的十进制值很不直观，要经过复杂的转换过程。如果把每一个十进制数字用字符编码存储，虽然 I/O 很方便，但一个字只能放两个数字，占用的存储量大。而采用一种 8421 制的 BCD 编码形式，可将十进制数字以压缩方式存储，而且读它的数值也很方便。

BCD 码码是取十进制数字的面值，即数字字符的 ASCll 码，取其低 4 位即可。以 4 位二进制位表达一个十进制数码,则一个字就可存储 4 个十进制数码，而且直接读出其十进制面值。例如：

1000 01 11 0110 0101

直接读出为十进制数 8765。可见 BCD 码和十进制数之间的转换是很容易的，只要用 4 位二进制直接代替每一位十进制。

用 BCD 码表示的十进制数，虽然也是“二进制的”，但它不满足二进制运算，因为二进位的加法进位是以 2 的幂为单位,而 BCD 码本质上是十进制数字的一种压缩存储形式，每 4 位相加应该是逢 10 进 1。事实上，有的计算机设计的十进制运算系统，就是对这种以 BCD 码表示的十进制数进行运算。

BCD 数的加法和四位无符号二进制数加法类似，但是，如结果超过 1001,则必须通过加 6 来修正。举例如下：

```
   4          0100
 + 5        + 0101
 ---        ------
   9          1001

   5          0101
 + 9        + 1001
 ---        ------
  14          1110
10+4        + 0110   一修正
            ------
            1 | 0100

   5          0101
 + 11       + 1011
 ---        ------
  16        1 | 0000
10+6        + 0110   一修正
            ------
            1 | 0110

   9          1001
 + 9        + 1001
 ---        ------
  18        1 | 0010
10+8        + 0110   一修正
            ------
            1 | 1000
```

值得注意的是，如两个 BCD 数相加时最初的二进制加法或者加 6 修正产生进位时，则向高一位产生一个进位。相类似的也可给出 BCD 码做减法的校正规则。

虽然 BCD 编码比二进制编码存储器利用效率低，但是在有大量十讲制数据输入 / 输出（I/O）的应用中，例如销售点终端，BCD 编码仍然是很有用的，因为它免除了在每次 I/O 操作时与二进制码之间的相互转换。许多计算机具有处理 BCD 数据的完整指令组，包括加法、减法及与二进制码的相互转换。其它一些计算机至少也有一个“十进制校正”指令，以使二进制运算能适用于十进制的数据。

1.5 补码运算系统

从本质上说，补码运算系统是无符号二进制数在同余的意义下(例如 mod 2^{16})所进行的二进制运算，至于这种运算的数值含义，或由硬件给出标志，或由软件给于解释。正因为补码表示的机器值与其代数值之间有固定的对应关系，而且又有良好的特性，所以当今大 多数计算机采用补码系统。

1.5.1 补码与真值之间的转换

按照补码的定义，给定真值求出它的补码，反之给定补码，也可求出真值。这里，由定义出发，推导出这两种转换的简便方法。

1.由真值求补码

给定真值 X=±X1X2…Xn，由补码定义，当 X≥0 时

[X]补码=0X1X2…Xn

当 X<0 时,由[X]补码=2^{n+1}+X,可以推导出

[X]补码=$1\overline{X_1}\overline{X_2}\cdots\overline{X_n}$

其中，$\overline{X_i}=1-X_i$，称为对 Xi 取反，即 0 变 1，1 变 0。

规则：正数的补码是其本身，负数的补码是把幅值各位取反，最高位之前冠 1（2^n）最后末尾+1(2^0)

例如：A=一 1101101 （令 n=7）

[幅值]取反=0010010

冠 1（2^n）=10010010

加 1（2^0）=10010011

这个结果，可以推导：当 X＞0，直接得到；当 X＜0 时：

[X]补码

$$
\begin{aligned}
&= 2^{n+1} + X = (2^{n+1} - 1) + X + 1 \\
&= 2^n + \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} X_{n-i} \bullet 2^i + 1 \\
&= 2^n + \sum_{i=0}^{n-1} (1 - X_{n-i}) 2^i + 1 \\
&= 1\overline{X_1}\,\overline{X_2} \cdots \overline{X_n} + 1
\end{aligned}
$$

2．由补码求直值

在 n＋1 位机器码中，已知 y=[X]补码=y0y1…yn，由定义，当 y0=0 时，真值 X=y1y2…yn；当 y0=1 时，真值

$X = -(\overline{y_1}\,\overline{y_2} \cdots \overline{y_n} + 1)$。

规则：补码的符号位 y0=0，真值 X 就是机器码本身，为正数。y0=1，真值 X 是负数，其幅值是补码的各位取反，末位再加 1（2^0）。

例如，令 n=7,y=10010011，因最高位是 1，真值 X 是负数，则各码位取反后再加 1 值以负号“-”表示，即：

X=-（01101100＋1）=-01101101

1．5．2 取反与取补运算

取反与取补都是对机器码的一种操作。假定有 n＋1 位机器码 y=y1y2…yn，取反操是对 y 的各位取反，

$\overline{y_i} = 1 - y_i$ (i=0,1,…,n)

记为：$[y]\text{取反} = \overline{y_0}\,\overline{y_1} \cdots \overline{y_n}$

或者记为：[y]取反=$(2^{n+1}-1)-y$

取补操作是 y 取反后，再加 1（2^0），记为[y]取补,即

[y]取补=[y]取反+1=$2^{n+1}-y=\overline{y_0}\,\overline{y_1} \cdots \overline{y_n} + 1$

取反、取补又称为求反、求补。

取反、取补的数值意义

[X]补码在经过取反或取补操作后，和真值的关系为：

([X]补码)取反=[-X-1]补码

([X]补码)取补=[-X]补码

根据取反、取补及补码的定义，对这两个结论是可以证明.不妨我们证明其中一个,另一个读者自己证明。

已知：y=[X]补码=y0y1…yn

求证：([X]补码)取补=[-X]补码

证明：由取补操作定义，有：

([X]补码)取补=[y]取补=2^{n+1}-y=2^{n+1}-[X]补码 ①

对①式我们分两种情况:

(1)当 X≥0 时,[X]补码=X,则

①=2^{n+1}-X=2^{n+1}+(-X)

由于 X≥0,则-X<0,由补码定义：

[-X]补码 =2^{n+1}+(-X),即

当 X≥0,有([X]补码)取补=[-X]补码

②当 X<0,有[X]补码=2^{n+1}+X,代入①式得到:

①=2^{n+1}-(2^{n+1}+X)=-X

由于 X<0,则-X>0,[-X]补码=-X,有

X<0 时,([X]补码)取补=[-X]补码

证明完毕。

这个结论的数值意义是，对[X]补码进行一次求补操作,是将真值 X 变号,再以补码表示之.若计算机设置一条求补指令,就可对数值变号操作.

例如:X=-1=-0000001(2) 令 n=7

[X]补码=2^8+X=2^8-1=11111111

([X]补码)取补=[11111111]取反+1=00000001=[-X]补码

1.5．3 补码的加法运算

我们在补码定义的一节里，利用同余的概念得到这样一个结论：任意两数的补码之和等于该两数之和的补码。现在给出这个结论的证明。

已知：具有 n 位二进制位的任意两数 X、Y，且

$-2^n \leqslant X < 2^n, -2^n \leqslant Y < 2^n, -2^n \leqslant X+Y < 2^n$,

求证：[X]补码+[Y]补码=[X+Y]补码

证明：分四种情况予以证明

（1）当 X≥0,Y≥0，且 X＋Y≥0

根据补码定义:

[X]补码=X

[Y]补码=Y

[X]补码+[Y]补码=[X+Y]补码

（2）X≥0,Y＜0．有

[X]补码=X

[Y]补码=2^{n+1}+Y

[X]补码+[Y]补码=2^{n+1}+(X+Y)

因为 X、Y 异号，若 $X+Y \geqslant 0$,一定有 $X+Y < 2^n$,则

$2^{n+1}+(X+Y)=X+Y \pmod{2^{n+1}}$

故有 [X]补码+[Y]补码=X+Y=[X+Y]补码 $(\mathrm{mod}\ 2^{n+1})$

因为 X、Y 异号，若 $X+Y < 0$,一定有 $-2^n \leqslant X+Y < 0$,则

[X]补码+[Y]补码=[X+Y]补码

（3）X＜0，Y≥00，根据（2）有

[X]补码+[Y]补码=[X+Y]补码

（4）X<0，Y＜0。由补码定义

[X]补码=2^{n+1}+X

[Y]补码=2^{n+1}+Y

[X]补码+[Y]补码=2^{n+1}+2^{n+1}+(X+Y)

因为 X<0、Y<0,有 X+Y<0,则

2^{n+1}+(X+Y)=[X+Y]补码,则

[X]补码+[Y]补码=X+Y=[X+Y]补码 (mod 2^{n+1})

综合四种情况，结论证毕。

这个结论同时也告诉我们,补码系统下相加的结果是有条件的,即在 mod 2^{n+1} 意义下的正确的.

1・5・4 补码的减法运算

由求补原理,把减法式子变换如下：

[X-Y]补码=[X+(-Y)]补码

=[X]补码+[-Y]补码

=[X]补码+([Y]补码)取补

由此，我们可以把减法改为加补。例如：

令 n=7,A=+119=+$1110111_{(2)}$

B=+17=+$0010001_{(2)}$

欲作; A-B=119-17=102=+1100110

有; [A-B]补码=01100110

且看;[A]补码+([B]补码)取补=01110111+(00010001)取补

=01110111+11101111

=01100110 (mod 2^8)

=[A-B]补码

采用补码系统，加法和减法可以统一起来，从而简化计算机的设计。由于减法和加补的等价性，只要把减数取补操作后相加，不必分别去考虑被减数与减数的符号是同号还是异号，像加法一样，符号和数值一齐参加运算，非常方便。

1．5．5 溢出问题

补码系统的加法和减法实际上是 n＋1 位无符号数的加减，但是作为代数值的运算，就有数值范围的限制问题。所谓溢出是指代数溢出，即运算结果超出一 2^n 到十（2^n一 1）的范围。那么在补码系统中运算如何判别呢？我们可作如下分析：异号相加无溢出，同号相减无溢出。但在下列四种情况时，加、减法可能有溢出：①两正数相加；②两负数相加；③正数减负数；④负数减正数。我们可从下面两个角度来判别代数溢出：

（1）同号相加（①，②两种情况），如果结果的符号位和原来的符号相异，则加法溢出。异号相减（③，④两种情况），如果结果的符号和被减数的符号相异，则减法溢出。

（2）因减法和加补等价，只需考虑两个无符号数的相加。如果进入符号位的进位（次高位的进位）和从符号位产生的进位（最高位的进位）相异，则加法或减法的结果是溢出的。因为把减法变成加补，异号相减变成同号相加,则③，④两种情况归结为①，②两种。那么，两个正数相加最高位一定无进位，如果此时次高位有向符号位进位，则结果变成负数。两个负数相加，最高位一定有进位，如果次高位不向符号位进位，则结果变成正数。因此第（2）种角度判别溢出的方法和（1）是一致的。

如果计算机硬件设计中没有设置判别溢出的标志,则程序员可用软件方法给出判别。

第 2 章 COMET 计算机及机器语言

计算机是能够存储算法和数据结构并能执行程序的综合装置。计算机可以用电子器件构成的实际物理设备，就像人们通常说的硬件计算机。但计算机也可利用软件(在某些另外的计算机上运行的程序)等价地构造，称之为软件模拟的计算机。COMET 计算机是一台抽象计算机，它是由宿主机的部分软件和部分硬件所构成的一台虚拟计算机。

COMET 计算机虽为软件摸拟的计算机，但它和一般的硬件计算机有着相似的逻辑结构。值得说明的是 COMET 计算机并不是在宿主机之上附加的类似操作系统那样的逻辑计算机，它是在计算机的基本结构和操作方面和实际的物理计算机是异曲同工的。同 样，COMET 计算机有自己的机器语言，CASL 汇编语言是该机器语言的符号化，并且具有汇编语言的基本特点。汇编语言是在具体计算机的寄存器一级上的程序设计语言。

2.1 COMET 计算机结构

2．1．1 逻辑功能结构

计算机通常都是由三个子系统组成（图 2．1）：存储器．中央外理机和输入/输出子系统.存储器能够存放数据、指令及运算的中间结果。中央处理机（CPU）能够执行算术逻辑运算功能，并且根据不同指令所译出的代码控制它应该做什么操作，何时做这些操作，即控制整个计算机的运算顺序。计算机与人们 的通信是通过输入／输出装置，为此，计算机还配置各种外部设备，作为输入／输出信息载体。COMET 计算机的逻辑结构如图 2．2 所示。

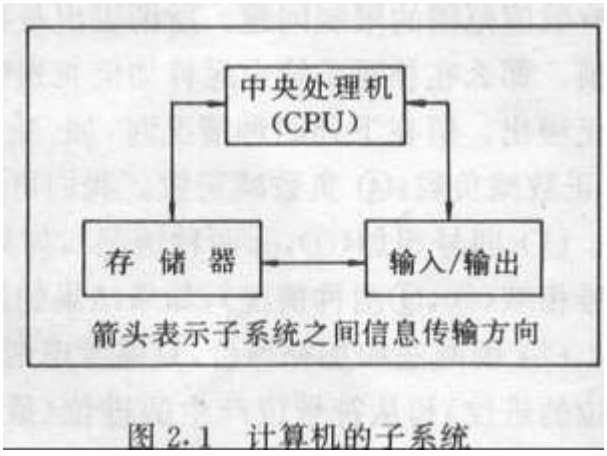


图 2.1 计算机的子系统

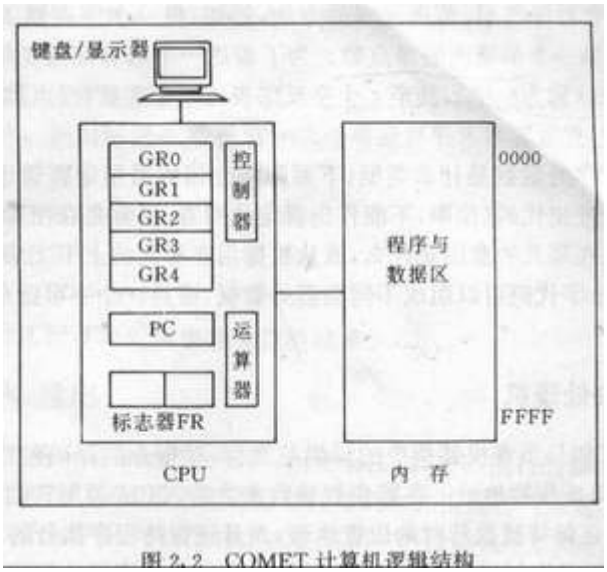


图 2.2 COMET 计算机逻辑结构

2．1．2 存储器结构

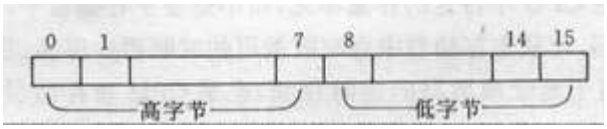
储存式程序的概念是迄今为止的所有计算机的基础,即指令和数据存放在同一个内存中。执行指令就是实现对数据的操作。既然程序和数据共存一个体内，如何区别它们？既然指令的执行，能够实现对数据的修改，又如何避免指令的执行而对指令的本身作了无意义的修改呢？这就给程序员提出了令人感兴趣的问题。

过去的计算机存储器是由称为磁芯的磁环组成的，每个磁环代表一位，由电磁转换，产生两种状态，一种状态取作为“0”，则另一种状态就被取作为“1”。现在的计算机大都是使用 MOS（Metel Oxide on Silicon）存储器，它是由具有成千上万微电子线路的片子所组成,它们中每一个单位在任何时候可以被打开或关闭，可把“打开”和“关闭”这样两种状态取作“1”和“0”。采用 MOS 存储器可获得比磁蕊存储器更高的存取速度。一般从内存到中央处理机，传输一个操作数，最小平均时间都在 300 毫微秒以下(1 毫微秒是 10^{-9} 秒)。

COMET 计算机的存储器是《CASL 虚拟执行环境》的组成部分，它是在宿主机的主存内开辟的一个 64K 字组成的内存空间,也是指令执行可能访问的数据地址最大范围。它的物理结构如下所述:

存储体：主存容量为 65536 个字，字的号码称为地址，其地址的编号是 0000-FFFF(十六进制).显然,一个 64K 字的存储器空间需要 16 位二进制位来表示.

字: 是由 16 个二进制位代码组成一个字,每 1 个二进位有一个号码,自左至右按 0-15 编号,高位编号是 0,低位编号是 15,如下图示:



尽管 COMET 计算机是按字编址，但我们仍可以逻辑地把一个字划分为两个字节，即左边 0—7 位约定为高字节,右边 8-15 位约定为低字节，这和流行的大多数计算机约定是一致的。定义字节为字符处理提供方便。

因为每位可取 0 或 1 两个值之一,则一个以 16 位二进制位组成的字就有 2^{16}=65536 种取值的可能。字被用于存储有符号整数，其范围-32768-+32767，也可用于存储无符号整数，其范围是 0—65535 或许它就是一个地址码，也可用于存储两个字符的 ASCll 代码。COMET 计算机约定 I/O 时一个字只存放一个字符数据，即它的高 8 位为零，低 8 位为字符的 ASCll 编码的二进制代码。

同样，对于其它数据类型，可用逻辑的方法，例如，将一个字存储 4 个压缩十进制数（BCD 码），也可存储一个单精度的浮点数 .为了表达一个具有高精度和更大数值范围的数,有时以双字长(称为长字),甚至 4 个字长来表达一个实数(浮点数).然而,COMET 计算机仅为字操作,整数算术及逻辑运算提供一套指令.

存储器中一个字的数据是什么类型(下面即将介绍的通用寄存器也是存储一个字的代码),仅从
它的二进位代码(位串)不能作出肯定的回答.通常是在使用它之前,我们或者计算机要知道它放在那儿的意图是什么,或从数据在那儿的上下文来决定.应指出的是,存储器中的一个字代码可以
组成不同类型的数据,而且一个字可能是一条机器指令的译码.

2．1．3 中央处理机

中央处理机（CPU）负责机器指令的译码与执行，对指令的译码包括确定它履行的是什么操作，并且计算操作数地址。在操作数被取来之后，CPU 运用它的算术、逻辑部件完成这个操作，CPU 还保持被执行时的位置轨迹，而且还保持程序执行的其它状态信息。如果 CPU 执行的是程序控制指令，它可以根据状态信息实现控制的转移。如果 CPU 执行的是输入／输出指令测可以通过输入／输出端口实现与外部设备交换信息。

以下介绍的通用寄存器、指令计数器、标志器是和程序员编制程序直接有关的三个组成部分。

1．通用寄存器

COMET 计算机有 5 个通用寄存器 GRO—GR4 长为 16 位，各位编号和存储器单元一个字的各位编号相同，即自左至右编号为 0—15，最高位编号是 0，最低位编号是 15，如下图所示：

通用寄存器是在 CPU 中特定的存储单元，而不是在主存储器中。在寄存器中存取及修改数据所花在时间，比从主存储器中存取数据用的时间要少得多。它们在汇编语言程序设计中，方便地被用于地址或数据的临时存储，也是 CPU 和存储器之间信息交换的桥梁。像一般计算机一样，COMET 计算机在机器指令中给出通用寄存器的特定标识位。

通用寄存器 GRO—GR4 又可作为变址器，它们由机器指令中变址器标识位指定。其中，通用寄存器 GR4 又兼作堆栈指针 SP。

2．指令计数器 PC

指令计数器 PC 是程序运行时的指针，用以记载指令的位置。它指向即将执行或正在 执行的那条指令的第一个字的地址。当该条指令执行结束时，置入下一条指令的地址，即指令结束时，

(PC)+2=>PC

只有执行转移指令，且转移条件满足时，才在 PC 中置人转移地址。在执行一程序之初，由系统将该程序的启动地址置入 PC，以后就由程序自身运行来控制顺序。

3．标志器 FR

FR 是 COMET 计算机设置的一个只有 2 位的标志寄存器，用于记录程序执行时外某些结果状态信息.例如可记录算术,逻辑运算或移位后的结果是大于零,等于零或小于零等三种状态之一。通常是：

当操作结果大于零（＞0）时：FR=00

当操作结果等于零（=0）时：FR=01

当操作结果小于零（＜0）时：FR=10

对于两操作数进行比较 FR 的值也有类似的结果。

2．1．4 输入／输出

计算机的输入／输出是很复杂的操作,其复杂性比 CPU 执行的算术与逻辑操作要高得多。输入／输出其所以复杂，是因为它含有计算机和外围设备的通信，诸如磁盘、终端以及打印机等多种不同的设备，可以某种方式安装到一台计算机上。不同设备有不同的数据类型及转换格式,不同的设备有不同的控制方式及状态信息，以及输入／输出将在哪里完成特定的操作。

汇编语言是输入／输出操作最有效的程序设计语言,它可以直接访问输入／输出接口，获取状态信息，高效而精确地控制设备进行各种输入／输出操作。然而一般的计算机不让 用户进行这些旨在使用外围设备的各种直接操作。所以作这样限制，一方面是因为系统的安全性之需；另一方面是使用户不必界入输入／输出设备物理特性的细节，用户通常是按系统规定的调用方式，逻辑地去使用外围设备。

由于 CASL 汇编的 I／O 只定义了两条宏指令，它们用于字符串的 I／O，因此 COMET 的 I／O 设备只需要两个，即键盘输入和显示器输出.

2•2 COMET 计算机的指令结构

2•2•1 指令的一般意义

汇编语言的程序是由各种有意义的指令组成的。一条指令是人们要计算机执行某一个特定操作的一种指示。当指令以二进制代码存在于计算机之中，并由 CPU 译码与执行才有意义。

指令通常由操作功能和操作对象两部分组成。操作功能由指令的操作码部分所代表的意义，这在计算机设计时就确定指令操作的详细要求；操作对象一般是指操作数，它由指令的地址码部分所标志。

机器指令的集合就是一台计算机的语言，计算机唯一能懂得的语言，就是全部指令在计算机中的代码，人们只有通过机器语言才能与计算机对话。其它一切高级语言，甚至汇编语言，最终都要被转换成机器指令的代码，方能被计算机所执行。显然，不同的计算机，它们的机器语言也是不同的，因为 CPU 唯一决定计算机的根本特征。

有了机器指令，人们就可按照一定的意图（给出算法），在计算机内存放若干条指令代码（编制程序），计算机就按一定的控制顺序，把指令的代码组合译码，转换成实际的操作序列，完成人们给定的预期任务（运行程序）。

要有效地使用指令，首先要了解指令的结构，指令的意义，寻址格式的规律，才能灵活地、合理地使用指令.

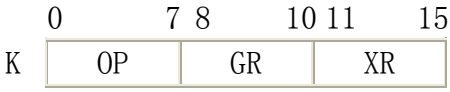
机器指令以二进制代码来表示，是计算机电子线路要求，不过，使用这种二进制代码(即使是八进 或十六制代码!)编制程序是很不方便的.实际使用的是与机器指令一一对应的符号指令，即用符号化的操作符号及地址符号来表达一条指令，这就是汇编指令。

用汇编指令来编制程序，既具有一定的可读性，又便于程序的增删及修改，给编程带来很大的方便。

2．2．2 指令的格式

COMET 计算机的指令代码是两个字长,即 32 位二进代码.一条指令以何种格式来表达操作功能与操作对象两部分呢？机器指令的复杂性,常常就在于表达操作数地址的多样性。COMET 计算机的寻址方式比较简单。在《CASL 虚拟执行环境》中对 COMET 计算机的目标指令给出如下定义：

指令格式：





其中，OP：8 位，表示指令的操作码。

GR：3 位，表示第一操作数或目的操作数的地址为通用寄存器 GRi=(i=0—4)。

XR：5 位，表示第二操作数的寻址方式。

ADR：16 位，位于双字长指令的第二字，可表示以下两种情况之一的汇编值：直接地址 A，变址的地址位移量 adr。

COMET 计算机的指令除表达操作功能是必备之外，它有三种方式来表达操作对象,这就是双操作数指令，单操作数指令，无操作数指令。

双操作数指令有两个操作对象，称第一操作数或目的操作数，以及第二操作数或源操作数。双操作数指令的第一操作数地址总是通用寄存器，通用寄存器可看作是 CPU 中特定的存储单元,通常以 GRi(i=0,1,2,3,4)来表示，它的目标代码对应 8—10 位,即 GR=0,1,2,3,4.第二操作数是由两种寻址方式之一来表达操作对象。双操作数指令执行后如果产生新的结果，则送到第一操作数地址 GRi 中，破坏了第一操作数原来的值,故第一操作数又称为目的操作数,第二操作数总是不变的,故又称为源操作数。但是,有一条存数指令 ST 是例外,按照指令的功能,把(GRi)送到第二操作数地址之中，破坏了原来的值。

单操作数指令的操作对象只有一个，通常是程序控制指令，如转移指令、转子指令等。无操作数指令是无操作对象部分,即操作码本身就能决定指令的操作结果,如 RET 指令。

2．2．3 指令的类型

按指令主要操作功能而言，COMET 计算机有 23 条指令，分为 7 类，现全部列出。根据学习要求，在以后章节中逐一作详细介绍。

（1）存储传送指令 3 条：

LD 取数 ST 存数 LEA 取地址

（2）算术与逻辑操作指令 5 条：

①算术操作：

ADD 加法 SUB 减法

②逻辑操作：

AND 逻辑乘 OR 逻辑加 EOR 按位加

（3）比较指令 2 条：

CPA 算术比较 CPL 逻辑比较

（4）移位操作指令 4 条：

SLA、SRA 算术左、右移 SLL、SRL 逻辑左、右移

（5）转移指令 5 条：

JMP 无条件转移 JPZ 大于等于转（≥0）

JZE 等于转（=0） JMI 小于转（＜0）

JNZ 不等于转（≠0）

（6）转子与返回指令 2 条；

CALL 调用子程序 RET 由子程序返回

（7）栈操作指令 2 条：

PUSH 进栈 POP 退栈

2．2．4 有效地址的确定

每种计算机都是根据其规模、速度、应用的不同，设计特有的一组寻址方式。指令系统的奇特性和复杂性，多半在于确定操作数有效地址的多样性。因此，熟悉寻址方式，乃是有效地使用指令的关键。COM 算机只有两种寻址方式,即直接地址型和变址型。根据这两种规则的约定求出操作数的有效地址。所谓有效地址就被访问的操作数的实际内存地址。

按指令格式中的 XR（第 11—15 位）区分两种寻址方式：当 XR=0 时直接地址方式，当 XR≠0 时变址寻址方式。

表 2．1 中约定变址型中的 j≠0 是 XR 中的***取值范围为 001—100，这与 GRO 不作变址器相一致的。汇编符号 A 和 adr 可以是 0—65535 中的十进制地址码，也可以是程序中定义的标号。而 32768—65535 的值也可以用负的十进制常数表示。下面以示例说明两种寻址方式是如何表达有效地址的。

表 2．1 COMET 计算机寻址方式

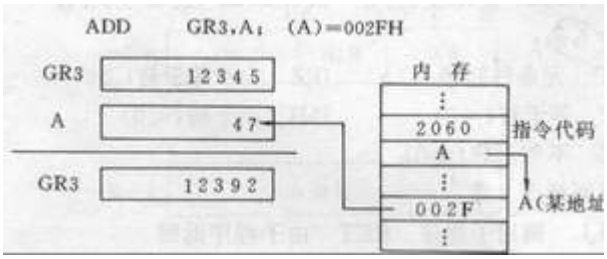
寻址方式	XR	汇编符号	有效地址 E
直接地址型	00000	A	指令第 2 字 ADR 是地址 A,即 E=A
变址型	00***	adr,GRj	E=AER+(GRj) j≠0

（1）直接地址寻址

以直接地址方式寻址，是允许指令用一个完整地址作为指令的一部分，被访问的地址紧跟在操作码之后的一个内存单元，如下图示：



当指令的代码 XR=00000 时，表示为直接地址方式寻址，在汇编指令中，被指定的地址可以是一个绝对地址码，也可以用一个名字 A（例如标号）直接书写，例如：



直接地址方式能方便地指定存储器空间中任何单元，程序中指定的直接地址 A，经过汇编时是单模块的汇编计数器值，再经过重定位后才是内存空间实际的地址值。

（2）变址寻址

变址方式寻址是指令执行时，用一个基地址和变址器形成内存的有效地址，即

E=ADR＋（GRj）（mod 2^{16}）

它的目标指令格式如下图示：

K	OP	GR	00***

K+1	ADR

ADR 是基地址的汇编值，紧跟在指令代码的下一个单元中。当指令代码中 XR=00***时，且***=j（j=1；2，3，4），即 j≠0，则表示为变址寻址方式，其中 j 以 3 位二进制代码表示，代表 GR1—GR4 中的编号。在汇编指令中，它用 adr，GRj 书写，其中 adr 可以是个绝对地址码，也可以是一个标号，它是变址方式中的基地址，也可称这是变址方式中的位移量.例如

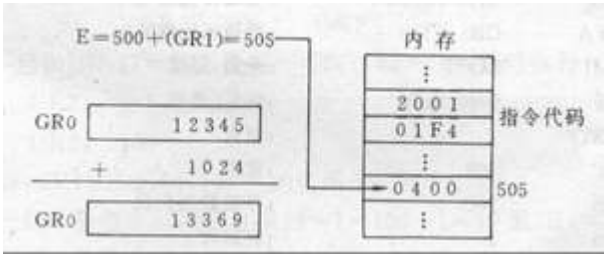
ADD GRO，500，GR1

假定（GR0）=12345

（GR1）=5

（505）=1024=0400H

由指令目标代码 XR=00001(2),它是变址方式,有



变址寻址的概念是和数组或数据表的处理相联系的。例如把一列数据排序，或涉及到多次访问一列相关的数据时，都需要用到数据表的处理。我们以一个数组求和的 PASCAL 程序为例：

```
SUM：＝0

FOR INDEX：＝1 TO 100 DO

SUM:=SUM+TABLE(INDEX)
```

如果用 COMET 计算机来完成这个求和任务时,在内存中用于数组 TABLE[1……100]的存储块是一个连续的存储区域，不妨假定数组元素 TABLE[i]为一个字长，则如下图示：

TABLE[1]	
TABLE[2]	
	……
TABLE[100]	

我们用两部分组成的地址标记来标识 TABLE 表中每一个元素,第一部分用标识符 TABLE，它对每一个元素都保持不变，第二部分为下标 INDEX，是可变的，对应每个元素分别取值在 1—100 之间。因此找数组中任一元素的有效地址，就依赖于这两部分地址，一个是静态的基地址 TABLE，另一个为下标值是动态的，它以一个变址器存放。因此数组中任一元素 i 的有效地址是这两部分之和，即

E=TABLE＋INDEX-1；INDEX=1，2，…，100

如果（GRj）= INDEX-1，则变址器 GRj 的初值是从 0 开始，这更适合于地址的编号，于是；

E=TABLE＋（GRj）

COMET 计算机提供变址寻址方式，就是适应这种需要，它是由计算机自动实现有效地址的计算，不必由程序员另外计算其值后，再填入指令第 2 字中。

利用变址寻址方式，得到一个数组求和的汇编语言程序如下：

```
        START
        LEA     GR0,0           ;求和的初态置 0
        ST      GR0,SUM
        LEA     GR1,0           ;变址器 GR1 的初值置 0
LOOP    ADD     GR0,TABLE,GR1;求和
        LEA     GR1,1,GR1       ;变址器值递增 1
        CPA     GR1,C100        ;已到末元素吗?
        JMI     LOOP            ;未到,继续
        ST      GR0,SUM         ;到了,送和
        EXIT                    ;结束
C100    DC      100             ;常数
TABLE   DS      100             ;存放数组元素
SUM     DS      1               ;存放和
        END
```

因为定义一个数组的第一个元素的地址是 TABLE 作为变址形式中的基地址，因此，变址器 GR1 的第 1 个值必须是 0。

2·3 COMET 计算机的指令系统

本节开始按功能分类介绍存储传送、算术逻辑操作、比较、移位及转移等 5 类指令，其余指令在以后章节中逐步介绍。全部指令汇集在附录 B，供编程查阅之用。

COMET 机指令具有通常计算机指令的基本特性，它的条数不多，易于学习。指令和各种寻址方式相结合，就构成丰富的机器语言。如果熟悉了寻址方式的使用，就会正确地理解 COMET 机指令的操作意义，并在进一步的编程中灵活地运用它们。

2·3·1 有储传送指令

1．装入指令 LD，存储指令 ST

LD 和 ST 是双操作数指令，实现寄存器与存储器之间的数据传送，其指令格式是：

(1)

LD A
GRi, adr,GRj; j≠0

有效地址 E=A/adr+(GRj)

功能:从有效地址 E 的存储单元中,读出代码送到寄存器 GRi 中,即

(E)=>GRi

(2)

ST A
GRi, adr,GRj; j≠0

同样，有效地址 E=A / adr＋（GRj）

功能：将通用寄存器 GRi 内容，存储到有效地址里的单元中，即

（GRi）=＞E

[例 2. 1] 已知操作意义，写出相应指令.

[解]①（100）=＞GR1，对应的指今为:

LD GR1，100

②（（GR2）＋100）=＞GRO，对有的指令为:

LD GRO，100，GR2

③（GR1）=A，（GR3）=＞A，对应的指令为：

ST GR3，0，GR1

[例 2. 2] 已知（GR1）=100，（100）=100，（99）=99，连续执行

LD GR2，一 1，GR1

ST GR2，100

两条指令后，GR1，GR2，99，100 内的值是什么？

[解]执行第 1 条指令时,E=（GR1）-1=100-1=99 故（E）=（99）=99=＞GR2。

执行第 2 条指令时，100 为直接地址，（GR2）=99，故 99=＞100。

结果：GR1 是变址器，内容不受影响，（GR1）=100，（GR2）=99

（99）=99 不变，（100）=99 破坏了原来的内容。

LD,ST 两条指令对标志器 FR 的值不受影响.

2. 装入地址指令 LEA

LEA 又称取地址指令，其指令格式是：

LEA　　A
GRi,　adr,GRj; j≠0

功能：将有效地址 E 本身作为操作数，装入到 GRi 中，即 E=＞GRi，其中 E=A/adr+（GRj）。

取地址指令 LEA 执行后，对 FR 的值有影响。LEA 是一条使用频度很高的指令，它常用于寄存器的赋值,寄存器的修改，寄存器间的传送。因为地址码也是一种数据，因此 LEA 常用在送一个立即数至寄存器中。

[例 2．3] 寄存器赋值

① LEA GR1，A

实现 A=＞GR1,A 可为一个标号，也可为一个地址码,在变址寻址方式中，对寄存器中赋一个地址的初值常用此形式.执行后,视 A 作为数值建立 FR 的值.

② LEA GR0，65535

实现 65535=＞GR0

这条指令执行后，因为 65535=（一 1）补码，是个负数，故 FR=10。

[例 2．4] 实现寄存器的修改

① LEA GR3，1，GR3，

实现（GR3）+1＝>GR3

② LEA GR2，-1，GR2

实现（GR2）-1=>GR2

这两条指令常用在循环变量的修改，对寄存器的修改可用任何操作数修改，其结果应为（GRi）（mod 2^{16}）

[例 2．5] 寄存器间的传送

① LEA GR1，0，GR2

实现（GR2）=>GR1

② LEA GRO，-48，GR1

实现（GR1）-48=>GRO，GR1 内容不变化。

2·3·2 算术与逻辑操作指令

全部算术与逻辑操作数指令,执行后都对标志器 FR 产生影响.

1. 双操作数的算术指令 ADD 和 SUB

指令格式：

$$\begin{matrix}\text{ADD}\\ \text{SUB}\end{matrix}\ \text{GRi},\ \begin{matrix}\text{A}\\ \text{adr,GRj;}\ j\neq 0\end{matrix}$$

其中 E=A / adr＋（GRj）

功能：（GRi）士（E）=>GRi

第一操作数地址总是用 GRi 指定寄存器 GRO—GR4 中任何一个，又作为放运算结果的目的地址。第二操作数地址 E 是按相应的寻址方式求出的有效地址。加法和减法运算均用补码操作。

（1）加法指令 ADD

操作意义：把第一操作数和第二操作数相加，其结果存入第一操作数的 GRi 中。第一操作数原来的内容破坏，第二操作数不变。

[例 2．6] 当（GRO）=32767，（A）=1，指令

ADD GRO，A

执行后，GRO 及 FR 值是什么？

[解] 两个正数相加，（GRO）=（GRO）＋（A）=32767+1=32768，而 32768=（一 2^{15}）补，它是一个负数，故：FR=10

[例 2．7] 当（GR1）=-2282，（300）=12931 指令

ADD GR1，300

执行后，GR1 及状态码的值是什么？

[解]（GR1）=（一 2282）补=63254 是负数，两数相加，

（GR1）=63254＋12931=76185=65536＋10649=10649（mod 2^{16}）

因为两异号相加，其和大于零，故 FR=00

（2）减法指令 SUB

操作意义：从第一操作数（被减数）中减去第二操作数（减数），其结果（差）送入第一操
作数的 GRi 中，第一操作数原来内容被破坏，第二操作数不变。

加法 ADD 和减法 SUB 的运算总是按机器码（补码）进行加减，按补码运算系统规则，其结果是正确的。但是若两数为有符号数运算，其结果要求满足范围：

-32768—+32767

否则溢出，具结果是不正确的。

[例 2．8] 当（GR0）=-2，（100）＝32767，执行指令 SUB GR0，100 结果如何?

[解] 按代数运算有:(GR0)=-2-32767=-32769<0 显然应该 FR=10.但是,结果的机器码是:

（一 32769）补码=2^{16}+(-2^{15}-1)=2^{15}-1=32767＞0

使得 FR=00。

出错的原因是因为结果超出有符号数的数值范围，导致溢出出错。

COMET 计算机没有设置数值运算的溢出标志.如果要求判定数值运算是否溢出，只能用软件的方法给予溢出的判定。

2．双操作数的逻辑指令 AND、OR 和 EOR

逻辑指令 AND,OR 和 EOR 的指令格式与双操作数的算术指令格式相仿，它们的第二操作数有效地址 E,或为直接地址,或由变址方式形成,第一操作数也是以寄存器 GRi 存放操作结果.

AND、OR 和 EOR 相应的逻辑操作后，决定 FR 的值是把操作结果（GRi）视为代数值，当:

结果 (GRi)>0 则 FR=00

(GRi)<0 则 FR=10

(GRi)=0 则 FR=01

以下分别说明 AND,OR 及 EOR 的功能特性

（1）逻辑乘指令 AND

逻辑乘指令 AND 实现对两操作数进行按位的逻辑“与”运莫。即相应“与”的两位全为 1，对应结果才为 1，否则为 0。逻辑乘操作常用于位运算，例如,要维护一个操作数的若干位不变，或将若干位置 0 时，用一个逻辑常数和它逻辑乘后即可解决。只要维持不变的对应位全为 1，逻辑乘后即可截下不变部分；要求置 0 的对应位全为 0，逻辑乘后即可将对应全置 0。

[例 2．9]

①截取 100 号单元内容的右半字送 GR2,用如下两条指令,即可实现取半字操作：

LEA GR2，255；255=FFH

AND GR2，100 ；（100）$_{8—15}$＝＞GR2

②用适当指令将 GR3 的第 12 位置 0，其它位不变。

只要将 GR3 和一个第 12 位为 0，其余各位为 1 的逻辑常数，做如下操作：

AND GR3，200

其中，（200）=FFF7H

（2）逻辑加指令 OR

逻辑加又称“或”运算,即两操作数的对应位中任一个为 1(或两个都为)时.则"或"的结果为 1，只有两对应位全为 0.“或”的结果才为 0.

逻辑加操作常用于位运算中部分位信息的叠加，例如，要使一个操作数部分位置 1（不管这些原来值是否为 1），只要用一相应位全为 1 的操作数和它进行逻辑加操作即可。

[例 2．10] 用适当指令，使 GRO 的第 11、12 位置 1，其它位信息不变。这只要用一个第 11、12 位为 1，其它为 0 的逻辑常数和 GRO 实行逻辑加即可：

OR GRO，H18

其中,H18 单元的内容是一个十六进制数 18 即可.

（3）异或指令 EOR

异或又称按位加，两操作数进行“异或”运算，是对应的两位值不相同时（一个为 1，另一个为 0）则相应位“异或”结果为 1，否则为 0（即两个同时为 1，或同时为 0）。

异或操作常用于位运算中，若要求一个操作数的部分位信息不变，而部分位取反（即 1 变 0，0 变 1）时，可用异或操作来实现。只要维持不变的这些位用“0”相“异或”，而要取反的那些位与“1”相“异或”即可。

[例 2．11]用适当的指令的序列，实现将 X 单元的内容取补并送回 X 单元。

我们已经知道，对一个字取补操作，就是将这个字对应的代数值变号。取补操作可用对各位取反,再末位加 1 来实现：

```
LEA     GR1,65535   ;65535=FFFFH
EOR     GR1,X       ;-(X)=>GR1,各位取反
LEA     GR1,1,GR1   ;末位加 1;
ST      GR1,X       ;取补操作后送回 X
```

应用异或的特性，还可用于判别两操作数是否相同。

[例 2．12]下列指令序列是一个顺序型的程序结构，当程序运行至 EXIT 结束后，问 5 个通用寄存器的十进制内容是什么？FR 的值是什么?

```
        START
        LEA     GR1,0
        LEA     GR4,N,GR1
        LD      GR1,N,GR1
        LEA     GR0,-1,GR1
        AND     GR0,-1,GR4
        LEA     GR2,0,GR1
        EOR     GR2,FFFF
        ADD     GR2,0,GR4
        ST      GR2,0,GR4
        LEA     GR3,2,GR2
        LD      GR4,FFFF,GR3
        EOR     GR4,FFFF
        EXIT
FFFF    DC      #FFFF
N       DC      256
        END
```

程序执行后，各寄存器的十进制值及 FR 值如下：

(GR0)=255
(GR1)=256
(GR2)=-1
(GR3)=1
(GR4)=0
FR=01

本例仅给出结果，至于这个结果如何得到，留给读者去做。

2．3．3 比较指令

COMET 计算机设置两条比较指令 CPA 和 CPL，它们都是双操作数指令，其指令格式为：

OP GRi,E

其中：OP 代表 CPA 或 CPL，为操作码；GRi 是存放第一操作数，为任何一通用寄存器;E 是第二操作数的有效地址，E=A 或 E=ADR＋（GRj）j≠0。

功能：将第一操作数与第二操作数，按照有符号数或无符号数的大、小或高、低进行比较。比较的方法是

(GRi)-(E)

的差,建立相应的 FR 值,而结果不送第一操作数.

比较指令常和转移指令配合使用,在分支或循环结构的程序中的判定或测试时出现。

1．算术比较指令 CPA

操作意义：视 GRi 及 E 中的代码为有符号数比较，象 SUB 指令那样作代数相减,依（GRi）-（E）所得的差建立 FR 的值，即;

当（GRi）＞（E）则 FR=00

当（GRi）=（E）则 FR=01

当（GRi）＜（E）则 FR=10

2．逻辑比较指令 CPL

操作意义：视 GRi 及 E 中的代码为无符号数比较，按如下条件，建立志器 FR 的值：

当（GRi）高于（E）则 FR=00

当（GRi）等于（E）则 FR=01

当（GRi）低于（E）则 FR=10

用高于，低于来理解两个无符号数之间的关系，或许更为贴切，它有利于和代数值的大小区分。为方便，有时仍用符号＞、=、＜表示高于、等于、小于。

[例 2．13] 假定（GRO）=-1，（ONE）=1，试述下列两条指令执行后 FR 取何值？

（1）CPA GRO，ONE

用作代数比较，显然 -1<1，则 FR=10

（2）CPL GRO，ONE

用作逻辑比较，视两操作数为无符号数，（一 1）补=65535，显然 65535 高于 1，故 FR=00

2．3．4 移位指令

COMET 计算机有算术移位 SLA、SRA，及逻辑移位 SLL、SRL 四条移位指令。它们都是双操作数指令，其指令格式为：

OP GRi，E

其中,OP 代表 4 条指令中任何一条的操作码;GRi 是通用寄存器，存放被移位的操作数；E 是指定要作移位操作的位数,E 或为直接地址值或为变址 adr＋（GRj）形成的地址值(j≠0).

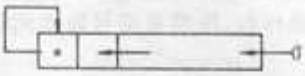
功能：将(GRi)按指定的有效地址 E 的值的本身(不是 E 的内容！),左移或右移 E 位,FR 值由最后一次移位的结果确定。即：

移位后的（GRi）>0 时, FR=00

移位后的（GRi）=0 时, FR=01

移位后的（GRi）<0 时, FR=10

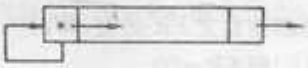
1. 算术左移 SLA



操作意义：除符号位，各位向左移 E 位，符号位不变，次高位丢失，低位移入 0。算术左移相当于

$(GRi) * 2^E => GRi$

2. 算术右移 SRA



操作意义：连同符号位，(GRi)向右移 E 位，符号位不变。算术右移相当于

$(GRi) \div 2^E => GRi$(整除)

[例 2.14] 若(GR0)=−14，执行下列指令后，GR0 及 FR 的值是什么？

```
SRA     GR0,  2
```

算术右移 2 位，相当于除 4，负数整除(向左取整)，则：(GR0)=−4，FR=10。

3. 逻辑左移 SLL



操作意义：(GRi)的 16 位代码一起向左移 E 位，最高位丢失，低位移入 0。

4. 逻辑右移 SRL



操作意义：(GRi)的 16 位代码一起向右移 E 位，最低位丢失，高位移入 0。

[例 2.15] 下列指令序列执行后,A 的内容起了什么变化?

```
LEA     GR0,1
SLL     GR0,11
EOR     GR0,A
ST      GR0,A
```

A 内容的第 4 位取反.

逻辑移位视(GRi)为逻辑代码,但它们对标志器 FR 产生影响,视(GRi)为移位之后的代数特性而决定 FR 的值.

2．3．5 转移指令

转移指令和转子指令都是程序控制指令，它们是以控制或改变程序执行的顺序为目的。本节主要介绍转移指令格式及使用特点，转子指令留待第五章的于程序设计中介绍。

转移指令是单操作数指令，它是以转移地址为操作对象（而不是访问操作数），其指令格式如下：

OP	A
	adr,GRj (GR0 不作变址器)

其中:OP 是任何一种转移指令的操作码，有效地址形成为：

E=A / adr＋（GRj）

COMET 计算机的转移指令是 64K 字全空间范围。

操作功能：无条件转移指令 JMP，总是执行 E=＞PC，而条件转移指令是满足其某种件，则 E=>PC，条件不满足,则顺序执行它的后继指令，即（PC）＋2=＞PC。

转移指令使程序控制多方向去执行成为可能,常在分支结构和循环结构的程序中出现。转移指令的操作意义明确，仅以表 2．2 列出各条指令的功能及转移条件。

表 2.2 转移指令表

操作码	功能	转移条件
JMP	无条件转移	必转
JPZ	大于等于转(≥0)/高于等于转	FR=00∨FR=01
JMI	小于转(<0)/低于转	FR=10
JNZ	不等于转(≠0)	FR=00∨FR=10
JZE	等于转(=0)	FR=01

由表 2．2 可知，条件转移指令能否实现转移要看标志器 FR 的值，因此在执行条件转移指令之前,要看它最靠近的那条对 FR 有影响的指令是什么.我们已经知道算术,逻辑运算指令,移位指令及算术比较指令 CPA 是视操作结果或操作对象为有符号数.这些指令执行后所建立的 FR 值,实行的条件转移可视为有符号条件转移.仅一条逻辑比较指令 CPL 是视操作数为无符号数，由 CPL 执行所建立的 FR 值，实行的条件转移可视为无符号条件转移。无符号条件转移通常用在比较地址的高低之后。另一方面，要实现一个正确的转移，注意选择何种指令来建立相应的 FR 值，以便有效地使用转移指令

[例 2．16]

①清除存储器从 3000 到 4000 单元的内容，使用如下指令序列：

```
        START
        LEA     GR1,3000
        LEA     GR0,0
LOOP    ST      GR0,0,GR1
        LEA     GR1,1,GR1
```

```
      CPA     GR1,AEND
      JMI     LOOP
      EXIT
AEND  DC      40001
```

这里使用算术比较指令 CPA 建立状态码，接下去执行 JMI 指令时，因为 GR1 取值自 3001 到 4000 都比 4001 小，即 FR=10，满足转移条件，循环继续，以致达到把 3000 到 4000 范围各单元清零的目的，直至循环终值（GR1）=4001 时，JMI 转移条件不满足，结束循环。由此，给定的指令序列是凑效的。

用同样的方法，再来看下面的问题发生了什么？

②把①中给定的比较常数改为：

```
AEND DC 40001
```

其余不变，能否达到清除从 3000 至 40001 范围的内存区域呢？我们先来跟踪程序的执行，一开始执行，使得（GR1）=3000（与①相同），接下去循环执行第 1 条指令和第 2 条指令，（GR1）=3001，仍然是算术比较指令 CPA，因为：

40001=65536—25535=[-25535]补码

显然，3001-（-25535）=28536＞0

仍用算术条件转移指令 JMI 来控制循环就不能凑效了。因为一进入循环，只清除一个 3000 号地址的内容，执行比较建立的 FR=00 不满足转移条件，循环终止。

如果用下面的指令序列，问题就全部解决了。

```
      START
      LEA     GR1,3000
      LEA     GR0,0
LOOP  ST      GR0,0,GR1
      LEA     GR1,1,GR1
      CPL     GR1,AEND
      JMI     LOOP
      EXIT
AEND  DC      40001
```

逻辑比较指令正是适应这类情况而设计的。读者还可运用它们来写出作双字长有符号整数比较的指令序列。

从实用的观点看，COMET 计算机所设置的条件转移指令是不完备的。例如，大于(＞0)转及小于等于（≤0）转没有。但是，我们可用组合的办法产生等效的功能。

[例 2．17] 写出指令序列，若 a＞b 转 FORTH，否则继续

```
        START
        LD      GR0,A   ;取 a
        CPA     GR0,B   ;和 b 比较
        JZE     NEXT    ;=继续
        JPZ     FORTH   ;>转移
NEXT    …               ;a≤b 均为继续
FORTH   …               ;a>b 时,执行的程序
                        段
```

2.4 例 题

[例 2．18] 当下列程序执行结束时，GR0 的内容是什么？

```
        START
BEGIN   LEA     GR0,6
        SUB     GR0,TEMP
        ST      GR0,TEMP
ZBR     JZE     BEGIN
        EOR     GR0,TEMP
        EXIT
TEMP    DC      6
        END
```

第一次执行到 ZBR 时有

（GR0）=（TEMP）=6-6=0=＞TEMP

结果为 0，因此转移到 BEGIN，继续执行使得：

（GR0）=6

结果非 0，因此顺序执行下条指令

EOR GR0，TEMP

因为（TEMP）=6，因此

（GR0）=0

[例 2．19] 在 DATA 单元内有代码 d，要求给出指令序列，能数出 d 中含有 1 的个数,其指令序列如下：

```
        START
        LEA     GR3,0       ;计数的初态为 0
```

```
        LS      GR2,DATA   ;代码 d=>GR2
REPEAT  LEA     GR2,0,GR2  ;建立状态码
        JZE     STOP       ;(GR2)=0,结束
        JPZ     SHIFT      ;最高位是 0,转
        LEA     GR3,1,GR3  ;最高位是 1,则
                           计数
SHIFT   SLL     GR3,1      ;左移 1 位,低位
                           补 0
        JMP     REPEAT     ;继续数 1
STOP    EXIT               ;数 1 结束
```

[例 2.20] 把 0 号至 99 号单元的全部内容，向后移动一个单元，其指令序列为;

```
        START
        LEA     GR2,100
MOVE    LEA     GR2,-1,GR2
        LD      GR0,0,GR2
        ST      GR0,1,GR2
        JNZ     MOVE
        EXIT
```

本例的传送区间覆盖问题，向后平移一般采用"到传送"方法，这样“搬家”既不要开辟新的区间，又不会破坏原来的内容。现在我们在上述程序中加一条指令

```
SLA GR0，0
```

变成如下指令序列：

```
        START
        LEA     GR2,100
MOVE    LEA     GR2,-1,GR2
        LD      GR0,0,GR2
        ST      GR0,1,GR2
        SLA     GR0,0
        JNZ     MOVE
        EXIT
```

试问读者，程序执行结束，又完成何种操作？

第 3 章 CASL 汇编语言及程序运行环境

本章介绍 CASL 汇编语言的语句格式及语法规则，给出源程序的书写格式及完整源程序的范例。CASL 汇编语言是一个虚拟的汇编语言，它的程序设计是在一个 CASL 虚拟执行环境中，完成其整个过程。因此，有必要介绍 CASL 虚拟执行环境的一些基本概念及其使用特点。各项具体操作命令可参考《CASL 虚拟执行环境用户手册》。

本章内容给那些使用汇编语言进行程序开发的读者是一个启示。对汇编语言有所了解的读者，可不必一一细读，即转向下一章的学习。

3.1 汇编语言

3．1．1 字符集

汇编语言的主要特点是符号化。符号由是特定的字符组成的，任何一个特定汇编语言有它允许的字符集。CASL 汇编语言允许下列字符的集合：

字母：大写字母 A—Z，若输入是小写字母，系统将小写和大写字母同样看待。

数字：0—99

分隔符：空格符（20H），制表符（09H），逗号“，”（2CH），分号“;”，行结束回车符 CR
（0DH）

指示符：十六进制指示符#,字符串定界符单引号们″’″,正负符号 +,-

3．1．2 标识符定义

标识符即符号名字。CASL 名字有两类：系统定义的名字，如指令操作符 ADD．伪指令名 START，宏指令名 IN 等;用户定义的名字由不超过 6 个字符组成,如 BEGIN,LOOP,X,L20 等。用户定义的标识
符规定为字母打头的字母数字组成,一个字符的名字,只能是字母.

用户名字用来定义标号（即符号地址）。系统约定，用户不得用系统定义过的名字来定义用户的名字。

凡别的模块要使用的名字必须是伪指令 START 的标号，全局名的值由连接 LINK 来确定。

3．1．3 常激

CASL 汇编语言中，允许下列四种常数：

（1）十进制数：由若干个 0—9 的数字组成。

（2）十六进制数：由若干个 0 一 9 的数字或字母 A—F 所组成的序列.为避免与标识符或十进制数混淆,凡在源程序中表示为十六讲制数的. 必须以“#”领首.

在源程序中书写的无论是十进制或十六进制数，经过汇编后，其机器代码均为 16 位（mod 2^{16}）的二进制代码；而在列表输出时均以十六进制形式表示。

（3）字符串常数:字符串常数是指用单引号“ ’ ”括起来的一个或多个字符，其值是各个字符对应的 ASCll 码。字符串常数一般是在伪指令 DC 中定义数据时出现。

（4）地址常数；地址常数是一个已定义的标号.标号所对应的地址码值，可作为一个字的二进制数据使用。

3·1·4 汇编语句

CASL 汇编语言的语句有三种类型，即 COMET 计算机的指令语句、伪指令语句和宏指令语句。语句的基本格式为：

[标号] △指令记忆符△操作数[；注释]＜CR＞

其中，标号和；注释分别加了方括号对“［”与“］”，说明这两个部分在一个语句中不是必要的。操作数部分可以是多个，它们之间用逗号“，”分隔，也可以是 1 个，甚至允许没有操作数。若为无操作数的指令语句，其操作功能由操作码规定。对于操作数的地址可以是定义的标号，也可以十进制整数或无符号数。注释部分要以分号“；”领首，也是它与操作数之间的分隔符，一个语句的注释是程序的说明性字段，在汇编时不产生任何目标成分，注 释可以用任何可印刷的字符。标号、指令记忆符及操作数之间都有 1 个分隔符“△”，△代表空格或制表符，语句行末有一个行结束符＜CR＞，它代表回车符（系统自动加上一个换行符）。今后书写源程序时，不作特别说明的，△和＜CR＞符都省略。

语句格式的书写：标准的汇编语句行各个字段之间为了整齐划一，在各分隔符之间可以加上制表符 TAB（09H），它使打印机或显示器从每个字段第 1 列开始到第 8 列为一组，不足 8 个字符的，自动以空格补充。每个字段位置开始列号如下：

列号	1	9	17	33
	标号	操作码	操作数	:注释

语句行可以是一个空行，即只有＜CR＞符，空行语句在汇编时不产生任何目标信息。语句行也可以是一个仅有注释的空行，注释可以从行首开始，只要用分号“；”领首即可。注释行可增加源程序的可读性，提倡多用，注释部分太长，可以分多行，但均要以“；”领首。

在结束本节时提醒读者注意：标号的结束符是空格或制表符，而不是许多汇编语言中的冒号“：”。

3.2 伪 指 令

伪指令是一种汇编命令,它是在汇编阶段中告诉汇编器如何进行汇编的要求与说明，控制汇编程序工作的一种指示，一般不产生可执行的机器指令代码。伪指令在汇编过程中就完成了它的使命，因此又称伪指令为非执行的指令。以下着重介绍常用的他指令。

3．2．1 定义数据伪指令 DC

定义字数据的伪指令 DC，其格式为：

DC n

其中数据 n 可为以下 4 种类型：

1．十进制数 d,d 为有符号数（-32768≤d≤＋32767）,d 也可用无符号数（0≤d≤65535）。

2．十六进制数 h，按约定十六进制数应写成 #h（0000≤h≤FFFF）。

3．标识符 a，a 是数据的符号化，一般是用来代表地址的。若 a 为本程序定义的标号，则汇编为地址码;a 若为在别的程序中定义的全局量，则由 LINK 确定 a 的值。

4. 字符串 C,按约定字符串 C 应加定界符″ ’ ″,因此单引号″ ’ ″不能作为字符串 C 中的一个字符来使用.汇编程序将每个字符的 ASCll 码从低地址到高地址，依次存放到每个字的低位字节部分，高位部分为 0。

3．2．2 定义空间伪指令 DS

DS 是为程序中保留指定长度改在储区域设置的，其格式为：

DS n

其中，n 可为十进制 d 或十六进制数 h （应写成 #h）,DS 语句的意义是从指定的地址起（一般是该语句的标号所代表的地址）保留 n 个字的空间，作为程序所使用的缓冲区。若 n 为 0 或缺省，该伪指令的标号值与下一个字地址相同.

3．2．3 程序人口伪指令 START

伪指令 START 的语句格式是：

[L] START [a]

其中，a 是地址码或本程序定义的标号，并指定它为本程序的启动地址。若 a 缺省，则指定本程序的第一个字的地址为启动地址。L 为 START 的语句标号，并由 START 定义 L 为全局量，可作其他程序的人口地址。L 的值与该程序的启动地址保持同值。

3．2．4 汇编结束伪指令 END

在每个源程序之后，必须写入 END 语句，它指示汇编程序，停止汇编在 END 之后的其它语句。

3.3 系统宏指令

在高级语言中提供有力的 I/O 语句，功能强而且使用方便。可是汇编语言中的人们操作很麻烦，即使像把内存中一个整数送到终端上显示这样简单的事情，都要一大堆操作步骤。

COMET 计算机没有设置 I/O 硬指令，因此 CASL 汇编语言中就不讨论复杂的 I/O 操作及中断程序设计。为了能够在程序中和外部世界进行信息交换，CASL 语言设置两条宏指令进行简单的字符串（记录）输入 / 输出。

为了更方便用户程序的数据交换或程序调试的需要，《CASL 虚拟执行环境》又为 CASL 语言扩充了两条宏指令进行十进制数的输入 / 输出。利用这些看起来像介于真实 I/O 指令和高级语言之间的宏指令就可方便进行数据交换。

以下分别介绍对系统宏指令的使用格式及功能。

3．3．1 输人字符串（记录）定指令 IN

格式：IN a1，a2

意义:从输入设备(键盘)上输入一个记录,即字符串,以回车结束输入.其中 a1 是存放字符串的首地址,每个字的低字节存放字符的 ASCII 码,高字节置 0;a2 是存放字符个数的地址,由系统将实际输入字符的个数填入 a2 内.

3．3．2 输出字符串（记录）定指令 OUT

格式：OUT a1，a2

意义：向输出设备--（显示器）上输出一个记录，即字符串。其中 a1 是存放字符串的首地址，每个字的低字节为字符的 ASC11 码，高字节为 0。a2 内存放的是输出字符串的长度，即字符个数。

3．3．3 十进制数输人定指令 DTOB

格式：DTOB

意义：由键盘上输入一个整数(-32768—+32767)或一个无符号整数(0—65535)，并转换成二进制数存入 GR0，以任何非十进制数数字的字符为结束。

3．3．4 机内数的十进制数输出定指令 BTOD

格式：BTOD

意义:将 GR0 内的二讲制数,以无符号的十进数（0—66535）向显示器上输出。

3．3．5 程序运行结束定指令 EXIT

格式；EXIT

意义：凡运行的程序，在程序的结束处，必须有该宏指令。

3·4 第一个程序

在本节前读者所见到的程序一般是不完整的。其目的是重点理解指令的功能及其使用特点。本节给出第一个完整的稷序，并通过它了解编程的全过程，以及在《CASL 虚拟执行环境》中，如何运用各阶段的软件工具，学习基本操作，知道它的全貌，以便于读者上机学习。

3．4．1 求解问题的定义

求解问题就是分析和明确题目的要求，给出求解问题的算法，以求得这个问题的正确解。

算法的一般意义是一个有穷规则的集合。是规定解决某一特定类型问题的操作步骤序列。简而言之，算法就是加工一个对象的手续。因此，确定一个问题的算法，必须是有穷性，即经过有限次运算，能得到结果的输出；其次，算法必须是确定性，即能用形式定义的计算机语言给予确切的描述，它不是像自然语言中所描写的"味道好极了"那样缺乏确定性。

一般地说，以某种计算机语言来描述一个算法，其表达形式就是一个程序。

[例 3.1] 从键盘上输入若干个不为零的整数，并求出其平均和在显示器上输出。

这个问题分为三步求解：

（1）分析数据类型及数据来源。本题要求被加工的的信息即数据是十进制整数，是从终端上随机输入不为零的数字字符串。进入计算机运算则要求数字串转化为二进制数，每一个十进制数要经过 10→2（十进制数转换为二进制数的简称），方能做求和运算。这里我们采用系统提供的宏指令 DTOB，实现把输入的数字串转换为二进制数。

（2）解题步序，即如何进行信息加工。

①利用宏指令 DTOB 把每个输入的十进制数转换成二进制数。

②把每个数存入缓冲区，并同时求和。由于要求平均和，故随之统计输入十进制数的个数。

③控制输入数据何时结束，这里采用输入一个“0”作为结束标志，实际上是以一个回车符，其 10→2 的值也为“0”且不计入个数。

④输入求和结束后即转入求平均和。求平均和应该作一次除法，但是 COMET 计算机没有设置除法指令，我们这里采用多次减法来替代除法。

⑤做除法时，首先要判定除数不为零。设置一个计数器（初始应为 0），当被除数大于等于除
数时，则做一次减法，同时计数器加 1，直到不够减为止，计数器的值即为商（整除），这个商就是求得的结果——平均和。

（3）输出结果。

为了能在计算机上观察所期望的结果，应尽可能地把计算结果显示出来，以检查算法的合理
性及正确性。有时在程序的出口处输出重要的提示信息，包括可能出错信息，使程序具有良好的可操作性。

这里计算的平均和是二进制数，输出是十进制的数字串（字符型）形式。2→10（二进制数转换
成十进制数简称）采用系统宏指令 BTOD 来实现.

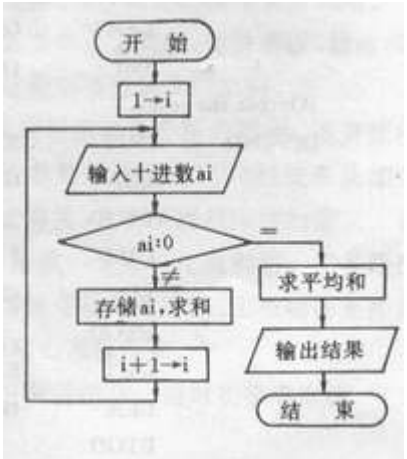


图 3.1 [例 3.1]的逻辑流程图

```
;Program for Example 3.1
;Get several no-zero inegers from the
;terminal, then calculate the average sum
       START BEGIN
BUFF   DS    20           ;data buffer
BEGIN  OUT   S0,L0
       LEA   GR0,0
       ST    GR0,SUM      ;Initilize SUM
       LEA   GR1,0        ;Index ptr initialized
;Input numbers into BUFF
LOOP   DTOB               ;Use DTOB to get a number
       CPA   GR0,ZERO     ;Skip when get the <CR>
       JZE   LOOPE
       ST    GR0,BUFF,GR1;Store the number
       LEA   GR1,1,GR1    ;Index forward ,count
       ADD   GR0,SUM      ;Get the sun
       ST    GR0,SUM
       JMP   LOOP
;Calculate the average sum, GR0/GR1-->GR2
LOOPE  LEA   GR2,0        ;Average sum initialized
       LD    GR0,SUM
       CPA   GR1,ZERO     ;Repeat if diviser is zero
       JZE   BEGIN
       ST    GR1,DVS      ;Do DIV using SUB
DIV    SUB   GR0,DVS
       JMI   DIVEND       ;Test if work should stop
       LEA   GR2,1,GR2    ;Modify the result
       JMP   DIV
;Output the result
DIVEND OUT   S1,L1        ;Output 'CIUNTER='
```

```
        LEA    GR0,0,GR1
        BTOD               ;Output the counter
        OUT    S2,L2       ;Output 'SUM='
        LD     GR0,SUM
        BTOD               ;Output the sum
        OUT    S3,L3       ;Output 'AVERAGE='
        LEA    GR0,0,GR2
        BTOD               ;Output the average
        EXIT
S0      DC     13,10,'INPUT NUMBERS:'
L0      DC     16
S1      DC     'COUNTER='
L1      DC     8
S2      DC     'SUM='
L2      DC     4
S3      DC     'AVERAGE='
L3      DC     8
SUM     DC     0
ZERO    DC     0
DVS     DS     1
        END
```

以上解题的三步就是一般的输入——处理——输出过程。虽然问题是简单的，但我们是借题发挥，使读者了解一个全过程。求解实际问题的定义，算法设计可能要先形成一个恰当的数学模型，以及采用有效的数据结构，并采用合适的设计工具进行描述。一个问题求解也可能有几种算法，采用哪种算法为好，要看计算机环境及其可行性。其次是要考察实现这个算法的效率，即执行这个程序要求所耗费的时间短，占用的主存空间少等。

3．4．2 程序流程图

1．程序流程图的重要性。程序流程图也叫框图，是个有向图。流程图作为一种算法表达工具已普遍使用，用流程图来描述算法，具有直观、逻辑顺序清晰的特点。流程图也是程 序控制结构的写照，反复审查流程图可以考察算法的正确性，有了一个正确的流程图就可顺利地编制一个程序。

读者可能已熟悉 1—2 门高级语言，由于高级语言的可读性及概括性，加上所遇的问题简短，往往不写流程图也会编制程序。但在汇编语言中，流程图是很有必要的。学会并习惯使用程序流程图，可使你在程序设计方面训练有素，具有良好的程序设计风格。

流程图有利于程序的调试，读者应习惯去从流程图中去查问题，排除错误，修改程序。流程图也是软件文档的一部分，它是在以后的维护过程中不可缺少的资料。

2．程序流程图的画法。用流程图描述算法的粗细程度并无严格的规定。通常把粗框称为逻辑流程图，它是以常用的术语，用符号及一些参数描述变量之间的关系及加工顺序。逻辑流程图一般可不依赖于具体计算机，它接近算法，有利于理解问题的定义。有时为编制程序的需要，我们进一步把逻辑流程图细化，形成一个工程化流程图。它是经过地址的代真，消去参数．用符号地址、寄存器或变址器描述变量间的关系。工程框图更接近的于程序，有了正确的工程化流程图，就可毫不费力地写出源程序来。

粗细两种框图可视问题描述方便而定，有时还可两者结合。但对初学者来说，较为复杂的问题，编制程序时最好有粗、细两种框图。

3．程序流程图的符号。用图形来描述算法也是由来已久的方法，流行的各种图形及符号也不尽统一。图 3．2 给出的流程图符号是根据国家标准局公布的《信息处理——数据流程图、程序流程图、系统流程图、程序网络图和系统资源图的文件编制符号及约定》中的标准程序流程图符号，以便参照引用。

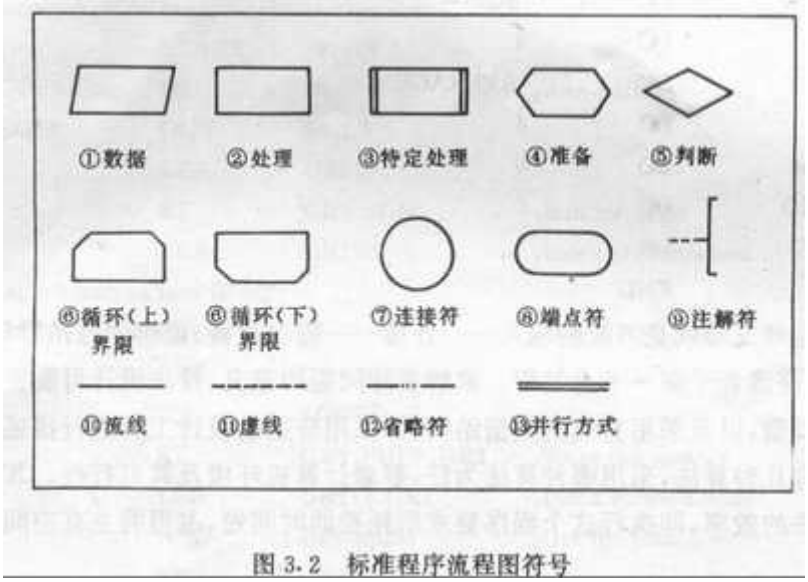


图 3.2 标准程序流程图符号

①数据，又称 1 们框，用平行四边形，标明数据名、来源、用途等。

②处理框，用矩形标明各种处理功能或特定操作。

③特定处理框，用双纵边的矩形标明在另外某处已得到详细说明的处理。例如子程序，模块的调用说明。

④准备，用六边形符号标明运算之前的预置工作。

⑤判定框，用菱形表示判定选择，它可以有多个出口，即从菱形引出若干流线，流线处应标有相应条件值。

⑥循环界限，用上下两个去角矩形组成，分别表示循环的开始和结束，均用同一循环标识名，并在开始或结束标明检验循环控制条件。在上下循环界限之间的应

是循环体的其他图形。描述循环也可不用循环上下界限图，直接以流线及其他符号的连接来描述。这两种方法各有优缺点。

⑦连接符，用圆形标明转出去处或何处转入的断点。同一转入或转出应是同名的符号或数字。连接符用于不在同一平面的流程或省略交叉的流线。

⑧端点符，用扁圆形标明一个流程的起点和终点。

其他符号，用到再加说明。

3．4．3 编写源程序

编程序是求解问题的关键部分，有了源程序，经过汇编，上机运行，才能最终取得解题的结果。

在一个源程序的首都，最好以注释的形式点出题解思想，重要工作单元寄存器或变址器的用途。在重要语句之后加注释，注明语句功能，在判定或转移的指令处需指明条件或去向，尤其是那些技巧性强的语句更要加短语作说明。加注释的程序具有良好的可读性，使程序便于交流，也便于调试及日后的维护。实际使用时，一般采用的是西文编辑，因此注释多数是用英文短语或汉语拼音比较方便。

提高程序的可读性是程序设计风格的要求，以后还要阐述结构化程序编制要求。读者可通过学习，力求在程序设计风格方面有较好的素养。

3．4．4 程序清单

程序清单通常是指一个程序经过汇编后列表文件。在下列清单中看到，左边是目标程序的代码，即机器语言形式（以十六进制格式书写），每条指令代码（包括数据）与存储地址相对应（一般是相对型地址）；在清单的右边是程序员编制的源程序，经过编辑器形成一个源文件。

程序清单具有目标代码与源代码的对应，目标地址与符号地址（标号及其位移量）的对应，可供了解执行程序（经过 LINK 重定位）的内存映象，用于程序的调试。

虽然程序清单是在正确汇编之后才建立的，但读者应了解，汇编时没有错的程序不等于程序没有错。汇编只能鉴别语法的合法性，而不能鉴别程序逻辑上的合法性。一般说，一个程序只有通过运行或跟踪调试，才能得到满意的结果。

```
File Name : T301.LIS
     ;Program for Example 3.1
     ;Get Several no-zero inegers from the
     ;terminal, then calculate the average sum
                    START BEGIN
0000           BUFF  DS    20          ;data buffer
0014           BEGIN OUT   SO,LO
```

```
0018 1200 0000        LEA   GR0,0
001A 1100 0082        ST    GR0,SUM       ;Initilize SUM
001C 1220 0000        LEA   GR1,0         ;Index ptr
                                          initialized
                ;Input numbers into BUFF
001E            LOOP  DTOB                ;Use DTOB to get a
                                          number
0020 4000 0083        CPA   GR0,ZERO      ;Skip when get the
                                          <CR>
0022 6400 000C        JZE   LOOPE
0024 1101 0000        ST    GR0,BUFF,GR1;Store the number
0026 1221 0001        LEA   GR1,1,GR1     ;Index forward ,count
0028 2000 0082        ADD   GR0,SUM       ;Get the sun
002A 1100 0082        ST    GR0,SUM
002C 6000 FFF2        JMP   LOOP
                ;Calculate the average sum, GR0/GR1-->GR2
002E 1240 0000  LOOPE LEA   GR2,0         ;Average sum
                                          initialized
0030 1000 0082        LD    GR0,SUM
0032 4020 0083        CPA   GR1,ZERO      ;Repeat if diviser is
                                          zero
0034 6400 FEE0        JZE   BEGIN
0036 1120 0084        ST    GR1,DVS       ;Do DIV using SUB
0038 2100 0084  DIV   SUB   GR0,DVS
003A 6200 0006        JMI   DIVEND        ;Test if work should
                                          stop
003C 1242 0001        LEA   GR2,1,GR2     ;Modify the result
003E 6000 FFFA        JMP   DIV
                ;Output the result
0040            DIVENDOUT   S1,L1         ;Output 'CIUNTER='
0044 1201 0000        LEA   GR0,0,GR1
0046                  BTOD                ;Output the counter
0048                  OUT   S2,L2         ;Output 'SUM='
004C 1000 0082        LD    GR0,SUM
004E                  BTOD                ;Output the sum
0050                  OUT   S3,L3         ;Output 'AVERAGE='
0054 1202 0000        LEA   GR0,0,GR2
0056                  BTOD                ;Output the average
0058                  EXIT
005A 000D 000A  S0    DC    13,10,'INPUT NUMBERS:'
     0049 004E
        …
     0053 003A
006A 0010       L0    DC    16
```

```
006B 0043 004F S1     DC    'COUNTER='
     0055 004E
     0054 0045
     0052 003D
0073 0008      L1     DC    8
0074 0053 0055 S2     DC    'SUM='
     004D 003D
0078 0004      L2     DC    4
0079 0041 0056 S3     DC    'AVERAGE='
     0045 0052
     0041 0047
     0045 003D
0081 0008      L3     DC    8
0082 0000      SUM    DC    0
0083 000       ZERO   DC    0
0084           DVS    DS    1
                      END
```

3·4．5 要点

程序设计风格可学的有许多方面，在此仅列出十条，但却是程序设计的重要方面.

（1）选择好的解题方法。

（2）画出逻辑结构清晰的流程图。

（3）编写程序力求通顺，不要过分使用技巧。

（4）用望文生意的标识符。

（5）加注释，增强可读性。

（6）插入提示标题，增强可操作性。

（7）注意程序的结构化。

（8）长的程序最好分块。

（9）程序要经过调试。

（10）欲修补“坏”程序，宁愿重新编写。

3.5 CASL 运行环境

CASL 虚拟执行环境，是抽象计算机汇编语言的运行环境，它是一个集成软件。CASL 汇编语言是 COMET 计算机上的汇编语言。COMET 是在另外计算机（宿主机）上模拟生成的一台“物理计算机”。CASL 软件系统具有编辑、汇编、连接、运行、调试等多个子功能系统。

本节旨在介绍 CASL 的使用及特点，而不是实现的方法。

3．5．1 用户界面

启动 CASL 之后，首先展示的如图 3．3 提供的系统主菜单，是一个多窗口的用户界面。用户可用移动“亮条光标”或键入菜单项目名字的首字母，即进入指定子系统操作。



图 3.3 系统主菜单

在进入下拉子菜单之后，可继续用光条移动选择命令。如进入 File 菜单，选择 Load 命令后，系统即提示你键入文件名，如果系统已存在你要的文件（默认为．ASM 的源文件）测立即在编辑窗口读出所需的源文件进行编辑。如果指定的文件不存在，则自动让你建立新文件。如果进入的是 Edit 菜单则系统把当前编辑缓冲区的源文件装入编辑窗口，让你使用编辑命令或功能键进行全屏幕编辑。

在进入任何一级子菜单时，如想退出当前屏幕返回先前的菜单，只要按 ESC 键，直到返回主菜单。要想退出 CASL 都必须由主菜单进入 File 子菜单，选择 Quit，返回到宿主机目录中的 DOS 提示符下。

3・5・2 一般操作流程

在 3．4 节中，从[例 3．1]的全过程可看出，我们在 CASL 上要经过哪些步骤，完成解题任务。图 3．4 反映这个操作的一般过程，以及各个阶段的任务和它们的结果。

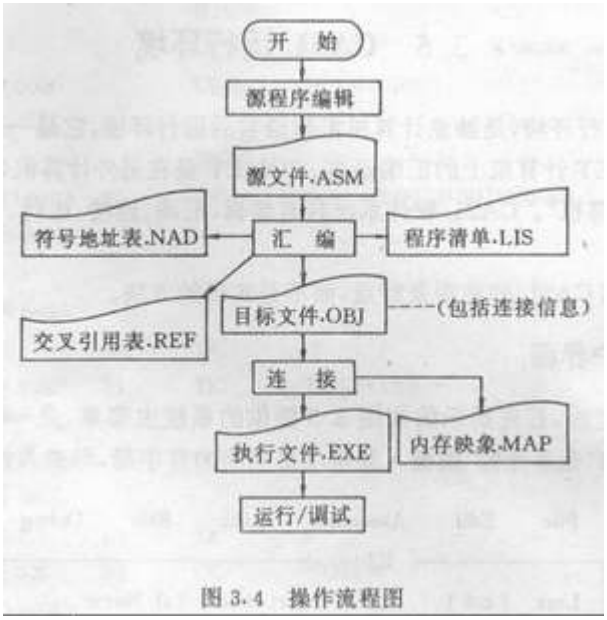


图 3.4 操作流程图

操作说明：

（1）利用主菜单可任意进入一个科目的操作。

（2）汇编是对当前编辑缓冲区存在的源文件进行。

（3）连接可对系统已建立的多目标文件进行。

（4）汇编、连接有错都不建立结果文件，一般都要重新修改源文件。

（5）CASL 所建立的各种文件，都符合 IBM-PC 系列的 DOS 文件约定。可选择 Os Shell 命令进入 DOS 操作文件。

3·5．3 汇编

在选择 Assemble 下拉子菜单后，可根据需要选择制表文件．LIS，符号地址对照文．NAD，符号引用文件．REF，最后把亮光条移到“Start Assemble”按下回车键即进行汇编。如果汇编没有语法性错误测汇编成功，并建立以下文件：

·供连接成可运行程序的目标文件．OBJ

·程序清单即列表文件．LIS（见 3．4．4）

·标号与地址（相对），对照文件．NAD

·反映标号定义行号及引用行号的交叉引用文件．REF

如果程序有语法性错误测在 Message 窗口报告出错信息并显示出错点的源代码行号，供修改源程序参考。有错的汇编测不建立新的文件，直到修改并汇编无错才为选择项指定建立相应文件。

如果在下拉子菜单中，未指定建立任何辅助文件，则光条停在"Start Assembler"，选择后仅生成一个目标文件．OBJ，通常都是选择这一种方式。

汇编是以 START 及 END 伪指令所包含的源程序段作为一汇编单位，建立一个独立的程序控制模块。一个程序可由几个控制模块组成，每个模块可独立成章，单独汇编，以后由连接程序 Link 把几个目标模块有机的连接起来，装配成运行的程序。

汇编子系统除了生成几个文件外，它还产生连接信息，供 Link 装配时用。CASL 的汇编系统采用两遍扫描；

・第一遍扫视：语句的合法性检查，建立用户符号地址表。

・第二遍扫视：生成每条指令的目标代码。

两遍扫视，对先使用后定义的符号确定其值（地址）提供可能，而且可避免用一遍扫视要经过大量回溯操作而降低汇编效率。

Assemble 具有以下职能：

（1）确定每条指令的起始地址及长度，建立源程序行和目标地址的索引，供源代码跟踪调试之用。

（2）建立各类符号索引表，提供连接信息。每个符号，用以下信息段建立索引：

・用字符串表示的符号名

・符号的地址值

・符号的属性是局部量，全局量还是外部量

・符号的引用范围

其中，符号的属性：

①局部量符号：本模块定义且本模块使用的，汇编成相对型地址，即每个程序段赋于 0 地址，计算出它的位移量，以后由 Link 装配成内存映象地址。

②全局量符号：本模块定义的且它模块可使用的。它是以 START 定义的标号，汇编 的暂定值，待 Link 连接时，根据连接信息，确定它的全局访问值。

③外部量符号：本模块使用而由外模块定义的全局符号，汇编时暂不报错，给一个待定值。外部符号的值由 Link 重定位时确定，若在 Link 连接模块中找不到其连接信息，则报错未定义符。

（3）确定程序段的长度及地址分配，并在第二遍扫视时确定每条指令的目标代码。

（4）处理伪指令。所有伪指令皆为系统定义的，查表并分别按其功能处理。

（5）宏指令处理。系统宏指令以内部中断替换，执行时进入系统调用。

3·5．4 连接

程序可由多个控制模块组成，各模块独立汇编时形成的目标模块一般都是相对地址。而且模块间符号的访问其值也未最后确定。因此要经过连接阶段，把这些目标模块加工成可执行的程序。

当选择 Link 下拉子菜单时，系统提示你键入目标文件名．例如，你要的运行程序是由 A．OBJ、B．OBJ、C．OBJ 三个模块组成的，则你可键入

A+B+C

此时的扩展名．OBJ 总是可缺省的。系统再提示你生成的可执行文件．EXE 不妨取第一个 A 为主模块，因此只要键入 A 即可，EXE 也是可缺省的。系统约定程序的启动地址总是以第一个模块由 START 定义的启动地址为整个程序的启动地址，故一般主模块放在第一个位置。如果一个程序仅由一个模块组成也必须经过上述的连接过程，只是在输入目标模块时，仅此一个罢了。

Link 具有以下职能：

（1）为各目标模块分配存储空间，确定浮动空间的分配方案。

（2）检查模块间的符号访问合法性。

（3）目标代码（程序及数据）地址，以及地址常数的重定位。

（4）给堆栈指针初始化，确定启动地址及内存映象，即符号访问的内存真地址对照
表．MAP。

（5）生成可执行的程序文件．EXE。

实际上 Link 除了接受的目标模块外，为形成一个可执行的程序，Assemble 还为 Link 提供以下的输入信息：

·程序控制模块名及其属性

• 符号的属性（局部，全局，外部等）

• 符号的汇编值

• 符号的交叉引用点及偏移量

由 Link 加工的目标模块，经过重定位，把相关的多模块组合成适于内存运行的完整程序。

3 • 5．5 运行 / 调试

一个程序经过编辑，汇编，连接得到可执行的程序．EXE。虽然这些过程是重要的，但对于一个问题的解仅是个准备过程。可以说没有经过运行 / 调试的程序是没有什么价值的。因此这最后一步也是最重要的一步，在这一步花去上述过程时间的总和也是理所当然的。一个具有一定规模（例如有 100 条指令）的程序，可能大部分的时间用在程序的调试上。经验告诉我们，没有一个完全排除了错误的程序，而只有错误未被发现的程序。

程序的调试过程就是对程序的测试，要分析程序的逻辑结构，给出相应的测试数据，让它充分暴露错误，以便修改程序。例如，用两边之和大于第三边判别一个三角形，当输入数据 2，7，5，程序运行结束，我们千万不要认为万事大吉了。虽然有 2+7＞5，但有三边值为 2，7,5 是不能构成三角形的。程序的测试是一门学问，其它科目还要专门研究它。

像其它计算机系统一样．提供有调试工具。CASL 系统提供的源代码跟踪的 Debug 是一个使用方便而有效的调试工具。当选择下拉子菜单“Debug”时回答一个需要运行的程序名（ • EXE 扩展名总是可缺省的）之后，屏幕上立即显示如图 3 • 5 那样的多窗口用户界面。

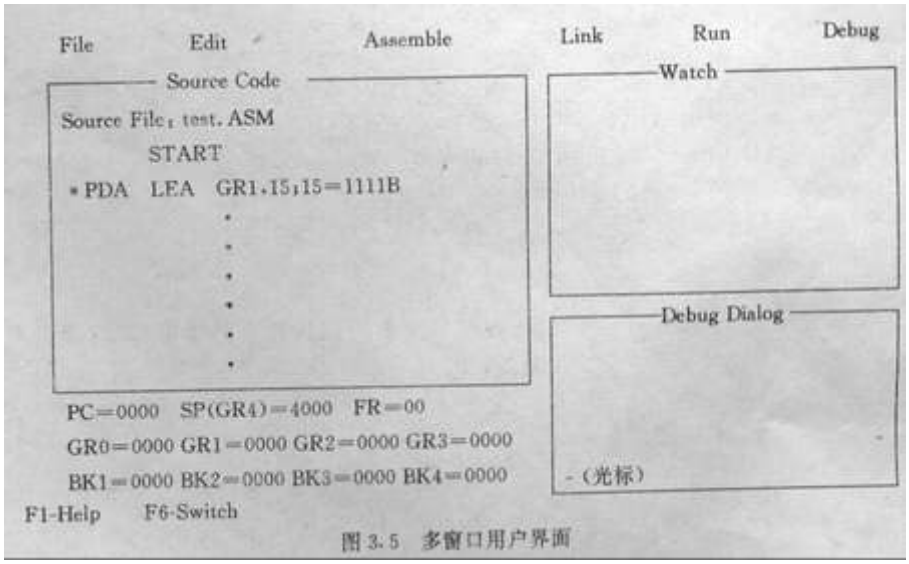


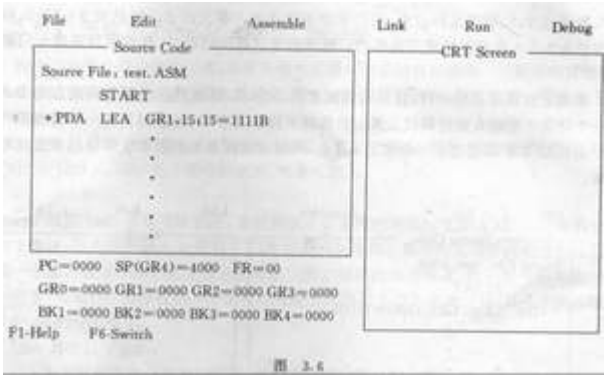
图 3.5 多窗口用户界面

用户调试界面有“Debug Dialog”窗口，用作显示输入的调试命令及系统的对话信息。“Source Code”窗口是源代码跟踪窗口，当用步进调试命令 T 时，有移动的星号“*”标记指出正在执行的那条指令，执行一条移动一条，非常直观。还有在辅助窗口显示每一条指令执行后的程序状态。有 PC 指示控制转移的地址石 P 指示的栈指针变化，还有标志器 FR 的值，通用寄存器的内容，一一显示在辅助窗口。

Debug 对程序的运行控制可以是单步运行，也可以是全速运行。为了动态观察程序运行的中间结果，Debug 有断点控制命令 BKi(i=1，2，3，4)。断点控制命令可在指定的地址 Ki（可用标号及其位移量表示）设置断点，当程序执行到指定的断点时，自动中止程序的执行。此时，一方面可在辅助窗口观察到当前的程序断点状态。也可进一步用 E 命令(R 命令)观察内存单元（寄存器）的内容或更新原来的内容，这既可修改数据又可修改指令。用设置断点的方法，能把错误局限在一定范围，避免错误波及而难以找到出错点。

经过使用各种调试命令，反复调试，找出程序的错误，再修改源程序，结果达到预定的要求，程序才算是基本上通过了。

程序中的输出可在 CRT 窗口显示结果，用 F6 切换，如图 3．6 所示。另外设有一个 WATCH 窗口，它可显示指定标号处的单元内容，并可跟踪运行，动态显示该单元内容的变化过程。



本节介绍了 CASL 系统的概貌，各子系统的使用特点及具体操作，请读者参阅《CASL 虚拟执行环境用户手册》（见附录 C）。

第 4 章 基本控制结构的程序设计

掌握了一种汇编语言的指令集，还只是学习汇编语言程序设计的第一步。下一步即从本章开始，学会使用这个指令集来构造程序，以及掌握程序设计中的一些基本技巧、方法，编制出适应具体要求的程序。

程序结构的设计是把算法转化为程序的准备。我们遵循结构化程序设计的思想，运用基本的控制结构或它们的组合，实现一些基本的程序设计要求。

程序设计是一门高度艺术化的科学，经过多年来的实践，人们已不断地发现和总结程序设计的基本原理和方法。在程序设计发展的历史中，Boehm 和 Jacopini 在 1966 年首先 用数学方法证明了程序设计语言中，只要有三种形式的控制结构，就足以表示出各式各样的其它形式结构。这三种基本控制结构是顺序型、选取型和重复型，如图 4．1 所示。

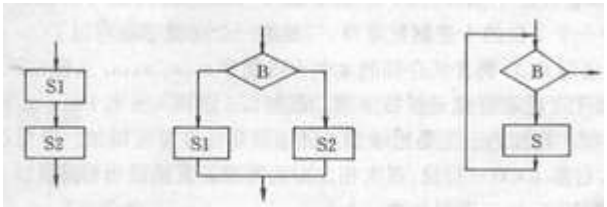


图 4.1 三种基本控制结构

我们可以从图 4．1 中看到，这三种基本控制结构有一个共同的特征，每种结构严格地只有一个入口和一个出口。如果组成程序的各个分结构之间都是如此简单的接口关系，那么就可相对独立地设计各个子结构，静态分析控制关系，以及验证它们的正确性。如果某一子结构需要修改，只要接口关系不变，就不会影响其它子结构乃至整个程序。因此具有这样结构的程序被认为是好的结构程序。

4.1 顺序型结构

顺序结构程序的特征是在一列指令中，没有转移性的指令。或者说，顺序型结构的程序，它的存储顺序和它的执行顺序是一致的。计算机在执行这个指令序列的第一条指令 后，接着指令计数器 PC 获得的是它的后继指令的地址，这样的顺序不会被指令执行而打乱，直到遇一条类似于停机指令执行而停止。

由于顺序结构的程序中没有转移指令观,则 PC 的内容就不会发生跳跃性的变化。如果粗略地把每一条指令的平均执行时间作一个单位时间，则 PC 的值是时间 t 的单调递增函数，如图 4．2 所示。

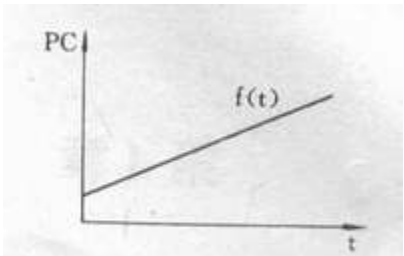
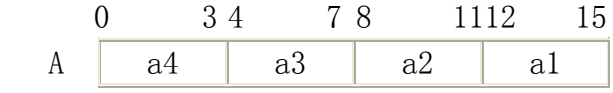


图 4.2 顺序执行时的 PC 变化

由此可知（PC）=f（t），即每经过一条指令平均执行时间 IC 的内容增加一条指令的平均长度。

任何复杂问题的求解，最终都要转化为算术或逻辑运算去完成，这些基本运算体就是一个表达式,用指令来描述各种表达式,就是一个一个顺序型结构的程序段。因此,又称顺序别结构的程序叫表达式程序.

[例 4.1] 已知存储字 A 中,存放有 4 个压缩十进制(BCD 码)数,形如:



求 B=a1+a2+a3+a4

解题思想：压缩十进制数，是指一个十进制数字串以 BCD 码方式存储的一种方法，因为一个十进制数字是以 ASC11 码形式存在的，一个 ASC11 码占 7 位，它的存储需要 1 个字节，为了节省存储空间，对每一个数字以 4 位二进制码就可以区别它们的数值（即 BCD 码），这样一个 4 位的十进制数字串，只要用一个存储字就可以了。

本例是对已用 BCD 码方式存储的 4 个十进制字 a1,a2,a3,a4。从低位到高位，每 4 位存储 1 个数字，每个 4 位正好就是该数字的二进制值。这样 a1+a2+a3+a4 就是一个 4 位的二进制代码相加。相加的方法是把每个 4 位移到最低位方可相加。即用逻辑常数截取 ai（i=1，2，3，4），右移 4*（i-1）位，逐次相加即为所求。直接给出程序。

;程序:BCD 码二进制加法

```
      START   PDA
A     DC      #A987
B     DS      1
PDA   LEA     GR1,15    ;#F
      AND     GR1,A
      ST      GR1,B     ;(B)=a1
      LEA     GR1,240   ;#F0
      AND     GR1,A
      SRL     GR1,4
      ADD     GR1,B
      ST      GR1,B
      LEA     GR1,3840  ;(B)=a1+a2
      AND     GR1,A     ;#F00
      SRL     GR1,8
      ADD     GR1,B
      ST      GR1,B     ;(B)=a1+a2+a3
      LEA     GR1,61440;#F000
      AND     GR1,A
      SRL     GR1,12    ;(B)=a1+a2+a3+a4
      ADD     GR1,B
```

```
        ST      GR1,B
        EXIT
        END
```

[例 4.2] 已知 a,b,c 为三个十进制数字字符,用它们构成一个十进制数 N=abc,试将 N 转换成二进制数存放在 GR0 中。

解题思想：根据十进制转换成二进制数的原则，要把 abc 转换成二进制数，实际上是求出表达式

$N=a\times10^2+b\times10^1+c\times10^0$

的值,这里 a,b,c 应是各个数字所对应的二进制值,十进制数字的二进制值可由：

a =’ a ’ –’ 0’

得到。其中 a 代表任一个十进制数字，而数字 0 的 ASCll 码是 48，只要每个数字的 ASCll 码减去一个基数 48 即可。其次，COMET 机没有提供乘法指令，但是乘 10 是可简单地用加法来实现。把上述表达式变换成

$N=(a\times10+b)\times10+c$

表达式经过这一变换，原来三个乘 10 运算变成两个乘 10 运算，而且这一次优化是很有意义的，既节省了运算时间，又缩短了程序的长度.算法可用图 4.3 所示的流程图描述.

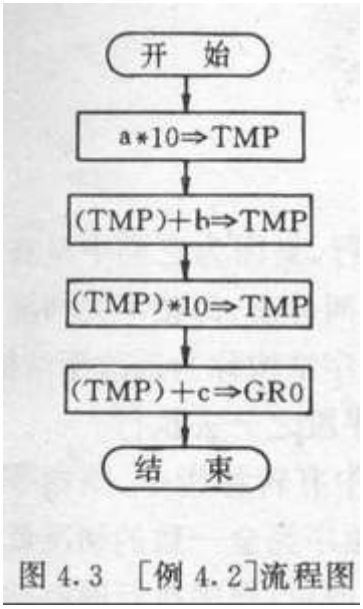


图 4.3 [例 4.2]流程图

;程序 4.2:三位数的 10→2

```
START   BEGIN
```

```
A       DC      a
B       DC      b
C       DC      c
TMP     DS      1
BEGIN   LD      GR0,A
        SUB     GR0,C48   ;a 的二进制值
        SLA     GR0,1     ;a 的二倍
        ST      GR0,TMP
        SLA     GR0,2     ;a 的八倍
        ADD     GR0,TMP
        ST      GR0,TMP   ;a 的十倍
        LD      GR0,B
        SUB     GR0,C48   ;b 的二进制值
        ADD     GR0,TMP   ;10a+b
        SLA     GR0,1     ;部分和的二倍
        ST      GR0,TMP
        SLA     GR0,2     ;部分和的八倍
        ADD     GR0,TMP
        ST      GR0,TMP   ;部分和的十倍
                          =>TMP
        LD      GR0,C
        SUB     GR0,C48   ;c 的二进制值
        ADD     GR0,TMP   ;结果=>GR0
        EXIT
C48     DC      48
        END
```

程序 4．2 中两次乘 10 运算的指令序列是相似的，这一思想在以后多位十进制数的 10→2，把这个“乘加”运算作为循环体，就可解决较为复杂的转换。

4.2 选取型结构

顺序型结构的程序所以能按指令存储的次序一条一条去执行，是因为它们中没有判定及转移指令存在。如果依据一个判定条件的真与假去分别处理两件事，即按不同情况去执行相应的程序段，就出现了分支的情况。这种有分支控制的程序结构称为选取型结构。

一般说，一个判定条件有两个分支，即由判定转移到两个分支程序段之一去执行。

选取型结构的程序中有转移指令，则指令的存储顺序和指令的执行顺序完全一致的情况就被

打破了。按照 4．1 节中的假定，程序执行时有转移指令,PC 的值就发生跳跃性变化，（PC）也不再是时间 t 的单调函数，如图 4．4 所示。

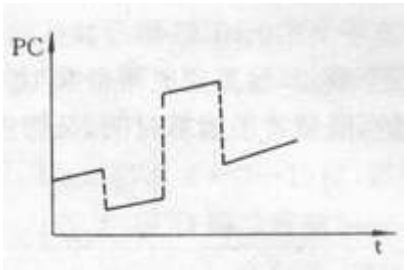


图 4.4 转移执行时的 PC 变化

多个判定条件产生多分支情况，但任何多分支都可转化为多个二分支情况来处理，所以首先
研究二分支结构。但是某些特定的多分支，例如类似 Pascal 的 CASE 语句所构成的多分支，常采用散转或跳跃表技术。

4．2．1 二分支结构

二分支结构一般如图 4．5 所示，即由一个判定条件，决定两个分支的控制流程。例如计算$\sqrt{x}$ ，就有两个分支的程序段，当判定 X 是负数则转移到出错处理的程序段，否则，转移到计算$\sqrt{x}$ 值的程序段。判定转移一般是由条件转移指令组成，而在条件转移指令之前一定有测试比较指令，或者是由那些能影响标志器 FR 的指令，执行后建立相应的 FR 值，以此来控制程序分支的去向。

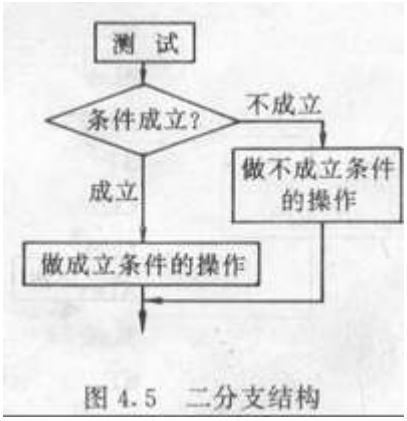


图 4.5 二分支结构

[例 4．3] 计算

$$y=\begin{cases} a+b+4c & x \geq 0 \\ a+b-4c & x<0 \end{cases}$$

解题思想:本例是一个二分支结构的程序,分支的条件由 x 决定,一支是完成 a+b+4c 的计算,另一支是完成 a+b-4c 的计算.但是程序在执行时,由 x 值决定只

执行其中一个分支.两个分支中都要计算 a+b 和 4*c,为了提高程序的效率,常常是把大家都要做的事情,提前（或推后）去统一处理。这种所谓"优化"，在汇编语言程序设计中,要学会处理.

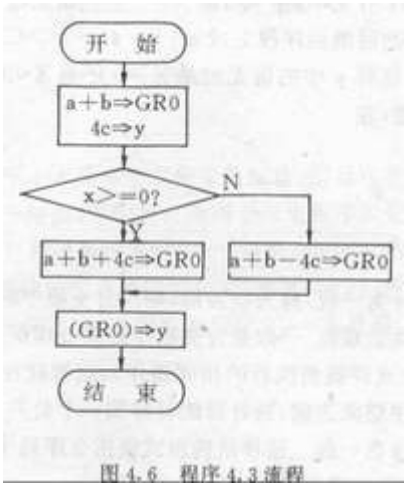


图 4.6 程序 4.3 流程

;程序 4.3:二分支处理

```
        START
        LD      GR0,A
        ADD     GR0,B     ;a+b=>GR0
        LD      GR1,C
        SLA     GR1,2     ;4c=>Y
        ST      GR1,Y
        LD      GR1,X
        AND     GR1,X     ;建立条件码
        JPZ     L0        ;X>=0,转 L0
        SUB     GR0,Y     ;否则,相减并转送
                          结果
        JMP     L1
L0      ADD     GR0,Y     ;相加
L1      ST      GR0,Y     ;送结果
        EXIT
A       DC      a         ;数据区域
B       DC      b
C       DC      c
X       DC      x
Y       DS      1
        END
```

程序 4. 3 中的指令

AND GR1,X

是 X 自身的逻辑乘，不改变 X 的值，但这条指令执行后的 FR 值，它的结果在 GR1 中且为
原来的 X，当 X≥0 则 FR=00 或 FR=01，否则 FR=10。它的下一条是条件转移指令

JPZ L0

实现控制分支，决定执行两个分支中的一个，即 X≥0 条件成立转向 L0 程序段完成 a 十 vb＋4c=＞GR0，否则执行它的后继程序段完成 a＋b—4c==＞GR0。不管哪一个分支做好，都要执行（GR0）=＞y，这样 y 中的值无论是 X≥0 还是 X＜0 取的值都是正确的，正符合题意要求。请读者注意，在

JPZ L0

指令的后继指令序列中有一条

JMP L1

指令是不可缺少的.

如果把测试和判定合并为一框,称为检验框,则二分支程序的一般情况见图 4.7。



图 4.7 二分支结构的一般情况

图 4．7 开始部分的公共处理框，一般是分支程序的预处理部分，有些操作在检验时或在检验后都要做（在逻辑上允许提前执行的相同操作），这样放在公共处理框可减少程序长度。各分支程序段在程序结束之前，最好都能转移到一个公共处理部分，使各分支处理后在某个地方控制再次结合在一起。这种结构形式使指令序列 1 和指令序列 2 都被嵌入在整个分主程序之中,对程序的理解和维护都带来好处。

例 [4．4] 编制程序，判别两数 x,y 符号的异同。若 x,y 同号则 GR0 置零，否则置 1。

解题思想：本题我们给出两种解法，分别为程序 4．4 及程序 4．4B。首先，按图 4．8 所描述的算法是一般常用的方法。按图 4．8 编制的程序 4．4 易读。

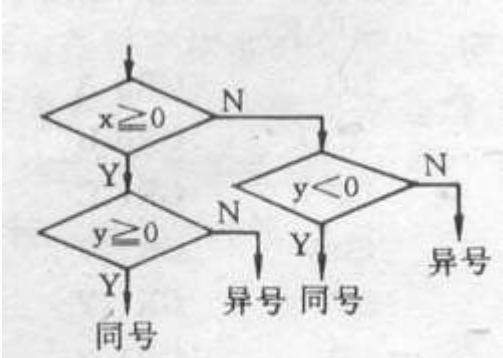


图 4.8 [例 4.4]算法之一

;程序 4·4：判别两数符号

```
        START
        LD      GR2,CY      ;y=>GR2
        LD      GR1,CX      ;x=>GR1
        LEA     GR1,0,GR1   ;测试 x
        JMI     XMI         ;X<0,转
        LEA     GR2,0,GR2   ;当 X>=0,测 y
        JPZ     ZERO        ;y>=0,,转同号处理
        JMP     ONE         ;否则当主<0,转 ONE,异号处理
XMI     LEA     GR2,0,GR2   ;当 X<0,测 y
        JMI     ZERO        ;y<0,转同号处理
ONE     LEA     GR0,1       ;异号处理
        JMP     STOP
        LEA     GR0,0       ;同号处理
ZERO    EXIT
STOP    DC      x
CX      DC      y
CY      END
```

从图 4．8 上看，程序 4．4 是多个二分支的组合，而且在处理 y≥0 及 y<0 的两个二分支结构中，算法描述十分相似，它们各自的两个分支程序段是一样的，所不同的是各自判定条件不同，一个是判定 y≥0，另一个是判定 y＜0。如果我们把算法稍加改变，当 X＜0 时，把 y 变个号，这样就可以把判定 y＜0 归并为判定 y≥0 去处理，如图 4．9 所示。

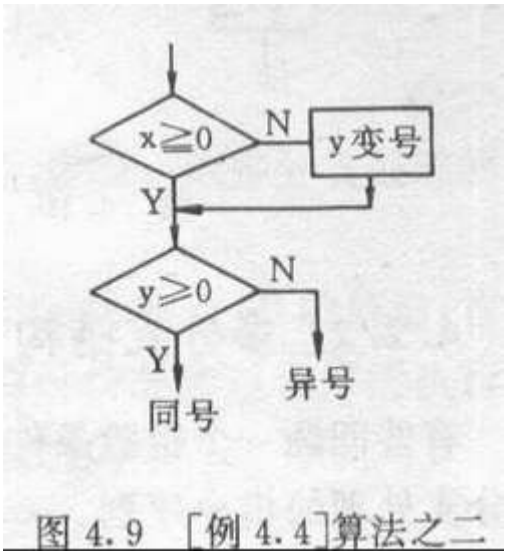


图 4.9 [例 4.4]算法之二

;程序 4.4B

```
        START
S       LEA     GR0,0           ;GR0 予置 0
        LD      GR2,CY          ;y=>GR2
        LD      GR1,CX          ;x=>GR1
        LEA     GR1,0,GR1       ;(GR1)=>GR1,测
                                试 x
        JPZ     LOG             ;x>=0,转
        EOR     GR2,FFFF        ;否则,y 变号
LOG     SLA     GR2,0           ;测试 y
        JPZ     STOP            ;y>=0,则
                                (GR0)=0
        ADD     GR0,ONE         ;y<=0,则
                                (GR0)=1
STOP    EXIT
CX      DC      x
CY      DC      y
ONE     DC      1
FFFF    DC      #FFFF
        END
```

图 4.9 给出的算法,在程序设计中称为程序结构的同构化.用同构化处理的程序可减少程序的长度。结构精巧。虽然同构化在现代程序设计方法学上并不强调，但在汇编程序设计中，适当作些训练，可以学到一点程序设计的技巧，还是有益的。

在编制二分支结构的程序时，可能出现两种情况：

（1）语句 if B then goto L1 else goto L2；结构形如图 4. 10。

（2）语句 if B then SI else S2；的结构形如图 4·11。

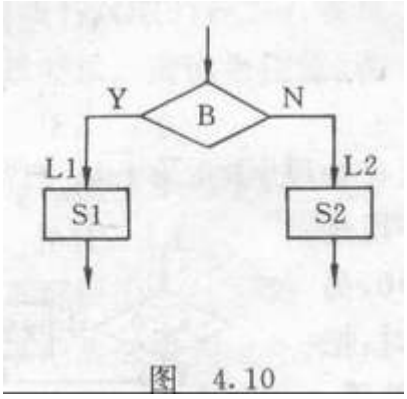


图 4.10

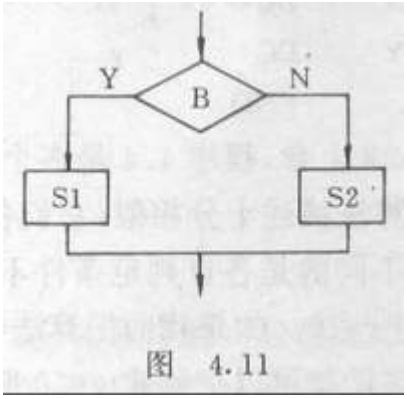


图 4.11

图 4．10 与图 4．11 中的 B 表示检验条件，L1 与 L2Z 表示语句标号,S1 与 S2Z 表示语句。第（1）种情况是缺乏程序的结构性，因为在执行 S1 或 S2 的路径后，能否再汇合，无然联系。这种多出口的结构给程序的分析，维护都带来困难。而第（2）种情形并非由语句标号来指定控制转向哪个语句，而是语句序列完整地被嵌入到这个二分的结构之中，这符合结构化程序设计的要求。由于以上我们给出的例子规模太小，读者还难以体会这一点。在一些大的程序中,有很多人参与编制程序,结构化的程序是倍受推崇的.

4·2·2 多分支结构

有些问题一个检验条件有多个选择出口，因此用一个条件转移是不够的，即程序有多个分支处理的指令序列。

[例 4．5] 求 F=Sign（x）。即

$$F=\begin{cases}1 & \text{当 } x>0\\ 0 & \text{当 } x=0\\ -1 & \text{当 } x<0\end{cases}$$

解题思想：据题意，根据 X 的不同情况 f 有 3 种取值可能，可用图 4．12 描述这个算法。这是一个多分支的结构,多分支在处理时可用多个二分支结构组合来现。

我们把它变换成图 4．13 的算法来编制程序。这里用两个条件判定，达到控制三个程序段的目的。

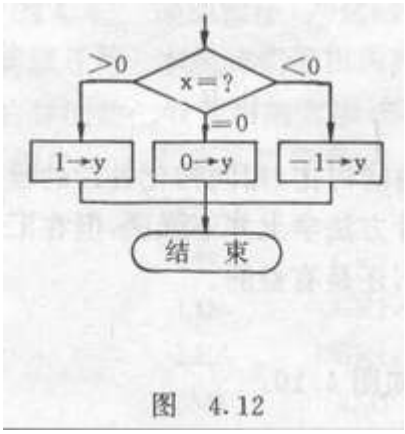


图 4.12

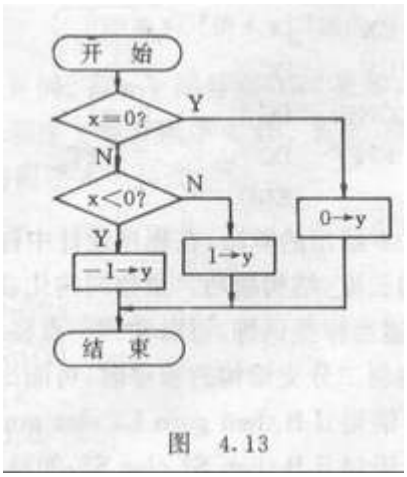


图 4.13

;程序 4.5:符号函数

```
        START
        LD      GR0,CX   ;取 x
        OR      GR0,CX   ;测试 x
        JZE     STOP     ;x=0 转
        JMI     JL       ;x<0 转
        LEA     GR0,1    ;x>的处理
        JMP     STOP
JL      LEA     GR0,-1   ;x<0 的处理
STOP    ST      GR0,F    ;送结果
        EXIT
CX      DC      x
F       DS      1
        END
```

这个程序虽然很短，但三个分支处理程序段的界面非常清楚，这是一般多分支处理的基本格式。

对于一些特定情况，可以运用一些技巧编制程序。本例给出如程序 4.5B 的解法，首先对结果预置 0（因为这是三个分支中的一种可能结果），以下两种情况通过程序的执行而分别得到结果。这种方法是三个处理的界面不清晰，但技巧性较强。

;程序 4.5B

```
        START  FXV
F       DS     1
CX      DS     1
ZERO    DC     0
FXV     LEA    GR2,0      ;0=>GR2
        LD     GR1,CX
        CPA    GR1,ZERO
        JZE    LF         ;x=0 then jump
        JPZ    L1         ;x>0 then jump
        LEA    GR2,-2     ;x<0
L1      LEA    GR2,1,GR2
LF      ST     GR2,F
        EXIT
        END
```

如果多分支的检验条件具有一定规则或可转化成有规则形式，那么对这类多分支结构可用一个有效的方法——散转方法——来实现。下面我们就来讨论这种方法。

4·2·3 散转结构

多分支中有种特定形式，例如 Pascal 中的 CASE 语句就是一种特殊多分支情况，现有

```
CASE   I    OF
1:     <语句序列 1>
2:     <语句序列 2>
  …      …
k:     <语句序列 k>
END
```

当 I（表达式）取值域 1，2，…，k（情况标号表）中的任何一个整数，则相应的一个语句序列得
到执行。情况标号表中标号常量可以不止 1 个（要求不重复），但它应与待选取的语句序列相关联。这个 CASE 语句的程序结构可用图 4.14 来描述。

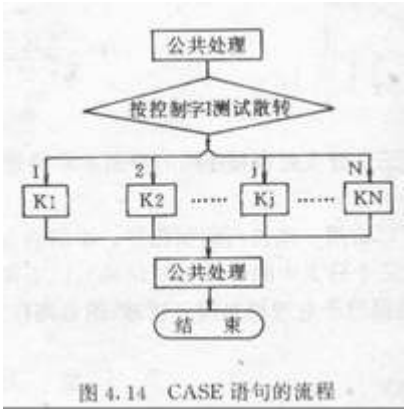


图 4.14 CASE 语句的流程

我们把＜语句序列 i＞对应的程序段取名为 Ki。,表达式 1 取值域整型量 1,2…,N 中的任意一个 i，则执行对应的程序段 Ki。这里把能控制执行某个程序段的 I 称为控制字。很显然，CASE 语句的程序是结构化的，即要执行的各个程序段都是嵌入在这个多分支结构中，这对分析或调试程序都十分有利。

CASE 语句的结构在汇编语言程序中可用以下三步来实现：

（1)用伪指令 DC 建立散转入口地址表 TABLE,假定各个程序段的入口地址为 K1，K2，…KN 则散转入口地址表为：

```
TABLE DC K1，K2，…KN
```

（2）将控制字互的值，置入变址器例如 GR1，用它来作为人口地址 Ki 的指针。

（3）构成散转格式，用变址器和 TABLE 相关联，实现散转，可用以下指令序列：

```
 LD    GR1,  I          ;取控制字
LEA    GR1,  -1,GR1     ;K1 的地址是 TABLE+0
 LD    GR3,  TABLE,GR1  ;取 Ki=>GR3
JMP    0,GR3            ;散转到 Ki
```

应注意的是，TABLE 表中的 K1，K2，…KN，应于 I 取值的顺序相一致，但并不要求各个程序段的位置顺序与此一致，即 K1，K2，…KN 的各程序段可以分布在任意内存空间，如果程序许可的话，总可以把控制传给某一与 1 相关的程序段。

[例 4．6] 编制计算下列函数的程序

y=

```
a+b-C          d=2
a-b-C          d=3
a ∧b-C         d=4
a+～b-C        d=5
a∨b-C          d=为其它
```

其中,a,b,c,d 均为已知代码.

解题思想：根据散转结构的分析，本例可用散转结构来编制程序。按照 d 的不同取值，有 5 个可能要处理的程序段，我们把 5 个程序段的入口地址，分别用标号

DADD，DSUB，DAND，DNOT，DOR

来标志，把它们依次存放在表首为 ENTRY 的缓冲区中，以构成散转入口地址表。其次，用控制字 d 来置变址器 GR3，为了使 d 取值与地址表中顺序相一致，我们要作个变换，用变换后的 i 作为散转控制字。

i=

0　当 d=2
1　当 d=3
2　当 d=4
3　当 d=5
4　当d=为其它

可用图 4.15 来描述程序的流程.

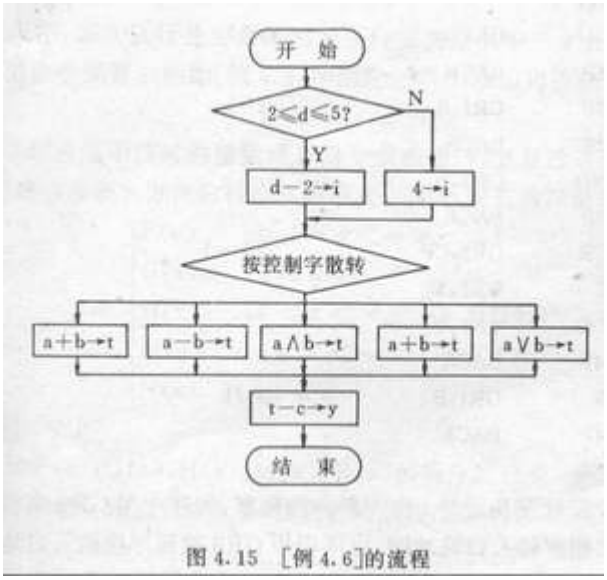


图 4.15 [例 4.6]的流程

;程序 4.6:散转处理

```
;y=a+b-c if d=2; a-b-c if d=3;
;a∧b-c if d=4; a+b-c if d=5;
;else a∨b-c
        START   BEGIN
A       DC      a
B       DC      b
C       DC      c
```

```
D       DC      d
Y       DS      1
C4      DC      4
CF      DC      -1
ENTRY DC DADD, DSUB,DAND, DNOT, DOR
BEGIN   LD      GR3,D           ;get control number d
        LD      GR1,A
        LD      GR2,B
        LEA     GR3,-2,GR3      ;tranfer control word 0 to
                                4
        CPL     GR3,C4          ;d is between 2 and 5?
        JMI     JUMP            ;yes
        LEA     GR3,4           ;else, set control word to
                                4
JUMP    LD      GR3,ENTRY,GR3 ;entry address to GR3
        JMP     0,GR3           ;jump
BACK    SUB     GR1,C
        ST      GR1,Y
        EXIT
DADD    ADD     GR1,B           ;a+b
        JMP     BACK
DSUB    SUB     GR1,B           ;a-b
        JMP     BACK
DAND    AND     GR1,B           ;a∧b
        JMP     BACK
DNOT    EOR     GR2,CF          ;a+～b
        ST      GR2,Y
        ADD     GR1,Y
        JMP     BACK
DOR     OR      GR1,B           ;a∨b
        JMP     BACK
        END
```

图 4·15 中的 d 经过变换后的 i 作为新的控制字，程序中用 GR3 来表示，由它作为散转的变址器，在取出相应的入口地址时，程序仍用 GR3 放程序段的人口地址，因为在取出入口地址后，原来的 i 已完成它的使命，又可用 GR3 作变址器实现转移。图中的 t 在程序中是 GR1,存放中间结果.

流程图中对 d 到 i 的变换，即

d=1,2,3,4,5 时,i=0,1,2,3

d 为其它时，i=4

按一般思路，我们可以用以下指令序列实现，即：

```
        CPA   GR3,TWO    ;d<2?,(TWO)=2
        JMI   OTHER      ;是,转
        CPA   GR3,SIX    ;否则,d>5,(SIX)=6
        JPZ   OTHER      ;是,转
        SUB   GR2,TWO    ;d-2→i
        JMP   JUMP
OTHER   LEA   GR3,4      ;其它情况,4→i
JUMP    …
```

然而在程序 4.6 中是用如下等价指令序列:

```
        SUB   GR3,-2,GR3
        CPL   GR3,C4
        JMI   JUMP
        LEA   GR3,4
JUMP    …
```

这里利用了这样一个性质，一切负数的机器码（无符号数）都高于正数的机器码巧妙地利用一条无符号比较指令 CPL 在实现对小于零及大于 3 的判定，同时测试两个条件。

分支结构的程序，每次执行是由具体判定条件而决定其中一个分支的程序段执行。同样，对于散转，是由选择器 I（例如[例 4．6]中的控制字 d）决定可控制转移的其中一个程序段执行。

如果控制字 I 取值是不连续的整数或者是分段取值，可以通过适当变换或者给出某种映照关系，使控制字改造为方便散转控制的形式。例如：现有分段函数

y=

F(X)	0<=X<=4
G(X)	4<X<=6
H(X)	6<X<=8(X 为整数)
M(X)	8<X<=12
N(X)	12<X

试编出计算 y 的程序。

假定计算函数 F（X），G（X），H（X），M（X），N（X）的程序段已有，且各程序段的入口地
址为 F，G，H，M,N。对 X 做适当的变换，使它成为散转控制字 I。这可以通过变换式

I=[(x+1)/2]

来实现，同时得到如下对应关系

当 0≤X≤4 时，I 取 0，1，2；

当 4＜X≤6 时，I 取 3；

当 6＜X≤8 时，I 取 4；

当 8＜X≤12 时，I 取 5，6；

当 12＜X，或 X＜0 时，令 I=7。

按照上述的对应关系，散转入口地址表可设计为：

```
LABEL DC F，F，F，G，H，M，M,N
```

因为 X 满足 0≤X≤4 时,I 取值为 0，1，2，所以设计的散转入中表，使 I 在取值为 0，1,2 时都能转向地址为 F 的程序段。我们可用如下程序段完成散转控制转移

```
      LD    GR1,X
      CPA   GR1,C13       ;(C13)=13
      JPZ   A
      LEA   GR1,1,GR1
      SRA   GR1,1         ;I=[(X+1)/2]
      JMP   JUMP
A     LEA   GR1,7         ;当 x>12,I=7
JUMP  LD    GR1,LABEL,GR1;实现散转
      JMP   0,GR1
```

散转技术还可以应用在散转的其它处理。例如“跳跃表计数”，假定有个计数表首为 TAB，表长为 N 个字，每个字作一个计数器。利用控制字 I 取值 i,控制第 i 个计数器计数，例如，每次加 1，若 i 满足：

1≤i≤N

执行如下操作：

i—1→i

（TAB＋i）＋1=＞TAB ＋ i

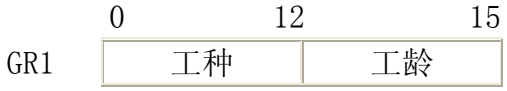
即可。由于控制字 I 取值是随机的。所以计数器 TAB 中各个计数器被跳跃式地计数，“跳跃表计数”由此而得名。使用指令序列

```
LD    GR1, I       ;取 i=>GR1
LEA   GR1,-1,GR1   ;i-1->i
LD    GR3,TAB,GR1 ;((GR1)+TAB)=>GR3
LEA   GR3,1,GR3    ;(GR3)+1=>GR3
ST    GR3,TAB,GR1 ;(GR3)=>(GR1)+TAB
```

就可实现第 i 个计数器计数。

散转技术应用在“散加”，“散取”等也是常见的。以下我们给出一个应用例.

[例 4·7] 由 GR1 给出的信息，编制计算职工的月工资程序.



其中:

$(GR1)_{0\sim1}$=

00 非技术工
01 技工 1 级
10 技工 2 级
11 技工 3 级

$(GR1)_{2\sim15}$=职工的工龄数(以月为单位)

月工资的计算标准：

（1）职工的月基本工资为：非技术工 80 元，技术工 100 元。

（2）职工的工龄超过 48 个月的，每 6 个月加 1 级津贴（不足 6 个月按 6 个月计），例如：49 月的加 1 级，55 个月的加 2 级，等等。

（3）每级津贴数为：

非技术工 5 元
技工 1 级 15 元
技工 2 级 20 元
技工 3 级 25 元

计算结果的月工资存 GR2.

解题思想，计算月工资信息由 GR1 的两部分信息位决定，第 0,1 位区别工种，首先中$(GR1)_{0\sim1}$决定非技术和技术工的月基本工资，即

$GR1)_{0\sim1}$=

=0　　非技术工 80 元
≠0　　技术工 100 元

其次，按级差津贴表 MONY，用“散加”每级津贴数。而计算津贴数需要区别超过 48 个月的，超过 48 个月的又要计算级数，因此要截下 GRI 原 2—15 位信息。

图 4．16 描述这个算法的流程。

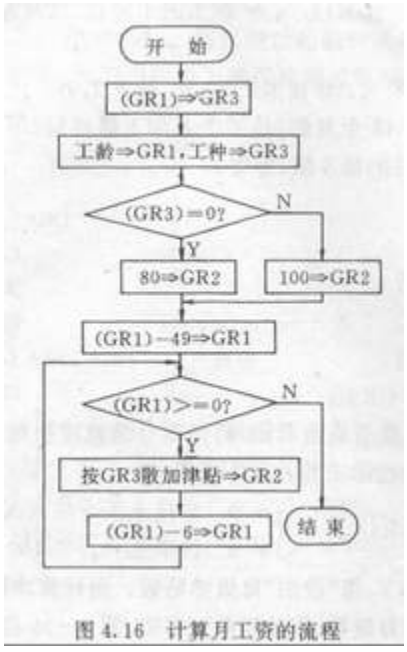


图 4.16　计算月工资的流程

；程序上人工资计算

```
        START
        LEA     GR3,0,GR1       ;准备信息分流
        AND     GR1,MSK         ;取工龄数=>GR1
        SRL     GR3,14          ;取工种=>GR3
        JZE     L80             ;非技术工, 转
        LEA     GR2,100         ;技术工的基本
                                工资
        JMP     GL              ;转 GL
L80     LEA     GR2,80          ;非技术工的基
                                本工资
GL      LEA     GR1,-49,GR1     ;工龄数减去 49
```

```
                                    个月
CMP     JMI     STOP                ;48 月内的,不加
                                    津贴
        ADD     GR2,MONY,GR3;散加津贴数
        SUB     GR1,-6,GR1  ;又是一级
        JMP     CMP
STOP    EXIT
MSK     DC      #3FFF       ;工龄屏蔽常数
MONY    DC      5           ;津贴级差表
        DC      15
        DC      20
        DC      25
        END
```

利用散加的技术，程序 4．7 编制得精巧，清晰，结构性好。否则，在计算月津贴时要用多个二分支，使得整个程序松散、冗长，不利于程序的分析与调试。

4·3 重复型结构

前面介绍的顺序结构及分支结构，在程序执行时，程序段最多只能执行一次。虽然在分支结构的程序中有转移指令（判定条件转移），但它是转移到从未执行过的程序段（某一分支）。本节开始介绍第三种基本控制结构即重复型结构。通常把重复型结构称为循环结构，循环结构从本质上讲也是一种分支结构，它也具有判定条件转移指令。不过，循环结构中的转移，是转移到曾经执行过的程序段（被称为循环体的程序段），这样就有可能重复测试和条件转移，形成“周而复始”的循环。能反复地执行一组指令是循环结构的重要特征，计算机的高速，正是利用这样一个重要特性，使它具有无与伦比的威力。

4·3·1 什么是循环结构

1．循环结构的建立。在数组处理的问题中，最常见的一个例子是对整型数组中所有元素的求和。例如数组 A，数组元素 A(i),用 PASCAL 语言求各元素和：

```
sum:=0

FOR i:=1 TO 100 DO

Sum:=A(i)+Sum
```

如何用汇编语言来描述这 100 个数求和呢？我们利用一长串加法指令去完成这数组 100 元素求
和，相加一个元素，就要一条加法指令。

如果真的要人们写上百余条指令，完成一个数组元素求和，那是一件令人乏味而又难以忍受的工作。试想，数组为一万个元素呢，计算机谁愿问津？

情况并非如此之糟糕！事实上，以表的形式存储及组织数据的本身，导致了一种程序结构类
型，那就是可以重复使用一组指令处理每项数据。因此，应用循环结构，这个问题可以变得很简单。图 4．17 是求数组 A 各元素之和的循环结构流程。

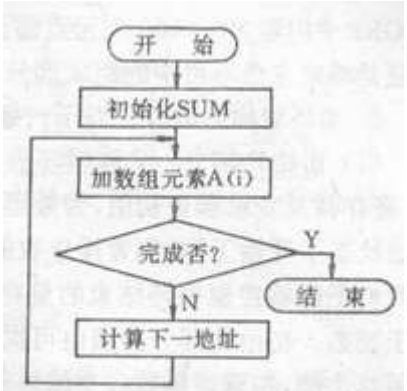


图 4.17 求数组 A 各元素之和

;数组 A 各元素之和程序如下:

```
        START   BEGIN
A       DC      A1,A2,…,A100;数组 A 的 100 个元素
BEGIN   LEA     GR1,A1        ;置变址器初态为 A
        LEA     GR2,100       ;计数为 100
        LEA     GR0,0         ;0=>GR0
LOOP    ADD     GR0,0,GR1     ;加 Ai
        LEA     GR1,1,GR1     ;地址递增
        LEA     GR2,-1,GR2    ;计数
        JNZ     LOOP          ;继续求和
        ST      GR0,SUM       ;送结果
        EXIT                  ;结束
SUM     DS      1             ;放和
        END
```

这是一个循环结构源程序，它的一个最显著的特征是,重复执行一组指令:

```
    ADD         GR0,0,GR1
    LEA         GR1,1,GR1
    LEA         GR2,-1,GR2
```

实现对不同数据项的求和操作.

作为循环体的一组指令是循环程序的核心部分,它每执行一次.导致地址的递增,使得求和访问数据的下一元素。可见，这种重复是在改变操作对象的条件下进行的，使每次重复是在新的意义下，选取新的操作数，产生新的结果。

构造一个循环结构的程序，仅有一个循环体还不够，在循环体工作之前，要给出第 1 个元素的地址（数据表的首址）。求和的初态要给出（作为求和的 GR0 要以零打底）。重复，不能无限制地重复，即如何控制循环何时结束要有具体措施。这里是用一个倒计数器 GR2，因为元素是 100 个，因此先置 100 为计数器初态，循环体执行一次，计数器减 1，直到 GR2 中内容为零，100 个元素都访问过了，正常结束。可见，修改计数器，判定循环是继续还是结束也是不可少的组成部分。

2.循环结构的成分。对于一般情况，一个循环结构程序，必须由以下四个部分组成：

（1）初始化部分：又称循环的预置部分，在循环体工作之前，要把有关变量，地址单元，寄存器及变址器置初值，为处理循环作好准备。初始化工作是保证循环程序在正确的初始状态下开始工作。如置操作数的首地址，指定循环体执行的次数，确定初始条件等。如果我们把能够控制循环结束的量称为控制变号的话，那么在初始化一般都要对控制变量赋于初态。初始化部分必须由可执行的指令组成，不能仅由伪指令替代，这是因为程序具有可复执性，在重新执行一个循环程序时，要把终态恢复到初态。

有时，我们把循环体中要做的相同操作，提前到循环体之外去做，也称为初始化部分。例如，循环体每次都要计算一个常量表达式，那可在循环体执行之前，先求出表达式值，再 参加循环体中运算。这种初始化称为循环的优化，优化可以提高整个程序的效率。

（2）循环体：这是整个循环程序的核心部分，它由需要重复操作的指令序列组成。循环体随求解问题的算法不同而异。因为它是重复执行部分，不仅要考虑它的正确性（当然要正确！），还要考虑到它的效率，因此这部分要精心编制。

（3）调整部分：为了保证循环体每次在新的意义下工作，每执行一次循环体、必须对操作数或操作数地址进行修改，为下一次重复作准备。同时控制变量亦应作相应的调整，为控制部分的条件判定作准备，例如统计循环次数，到一定次数时，控制部分根据计数器值判定，决定是否退出循环。通常的循环结构是调整和循环体工作同步,有时我们又把循环体和调整部分合在一起,统称循环工作部分。

（4）控制部分：为了能在正确的时机退出循环，要有出口测试。控制部分是用来控制循环程序是继续执行还是终止。控制部分是对修改过的控制变量进行测试或检验，若达到预定要求，则循环结束，否则循环继续进行。

有时，控制循环结束的判定条件是一个以上，例如是两个判定条件，就可能出现两个出口，编制程序时，应注意程序的结构性，使两个判定分支完全嵌入在程序之中。其次是，正确理解判定条件，是两个同时满足才结束循环，还是任意一个条件满足就结束循环。

对控制变量的测试，要注意边界状态的正确性。比如，要重复执行 n 次循环体，常常要检查第 1 次和第 n 次是否正确。这是因为采用不同的循环结构形式，其控制变量的初态或终态是有差异的。

以上所述的四部分是构成循环结构程序不可缺少的组成部分，特别是在嵌套循环程 序中，内循环的四个部分要完全嵌入在外循环的循环体中。

3．循环结构的模式。如上所述，循环是重复执行一列操作所构成的算法描述。如何控制循环体的重复执行，以及用什么方式控制循环的结束。大致有如图 4.18 所示的四种形式。

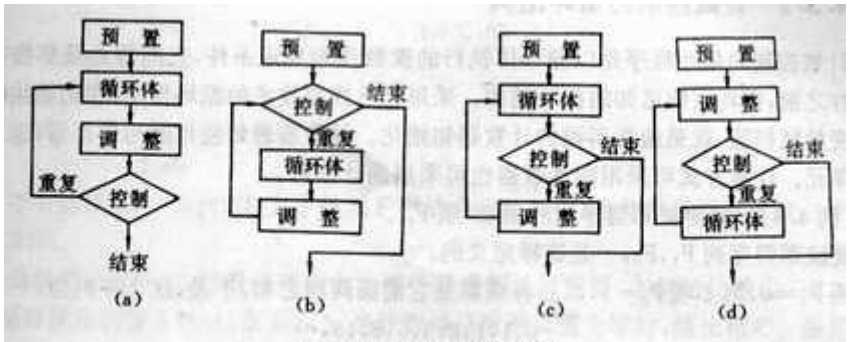


图 4.18 四种循环结构形式

图 4．18 分别由（a），（b），（c），（d）四个图组成，它反映循环结构程序所包含的四种形
式。这四种形式的主要区别在于先执行循环还是先进行判定的控制，（a），（c）反映前者，进入循环程序，循环体至少要执行一次；（b），（d）反映后者，循环体可以一次也不执行。其次，（a），（b）两种形式是循环体和调整部分的程序完全同步执行；（c），（d）两种形式的循环体和调整不完全同步，即两者执行重复的次数是不等的。采用哪种形式取决于具体同题而定，另一方面还要看程序员的习惯。循环结构的四种形式是汇编语言程序的特点，严格按
照这四种形式构造的程序，其结构性好，层次清晰，便于程序的分析和调试。初学者要注意，先形成程序结构，再编制程序，不可滥用 JMP 指令。

实际上，循环结构的四种形式是以下两种循环模式的细化。几乎所有语言的循环结构都遵循图 4．19 所示的两种模式。

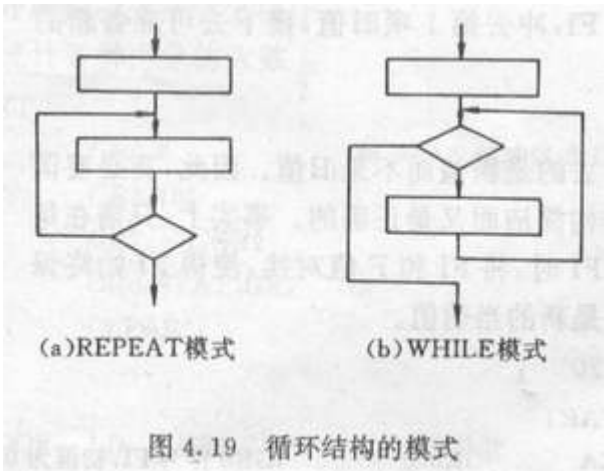


图 4.19 循环结构的模式

图 4．19 的模式(a)所示的控制结构的意义是，重复执行一列操作（包括循环体及其修改部分），直到循环结束的判定条件成立则退出循环，这种模式的循环体至少要执行一次；模式(b)所示的控制结构的意义是当循环的条件成立时，才重复执行一列操作，否则退出循环，这种模式的循环体可以一次也不执行。

循环结构的程序是多种多样的，特别是以三种基本控制结构连接或嵌套，就带来程序结构的复杂性。为了叙述方便，我们按控制变量的两种类型，（即计数型和条件型），通过典型的实例，分类阐述循环控制结构程序的设计方法。

4．3．2 计出控制的循环结构

计数控制的循环程序是以循环体执行的次数作为判定条件，它的特点是那些在循环体工作之前，循环次数已知的循环结构。采用计数控制技术的循环程序，在初始化时要对控制变量赋初值。就是通常所说的计数器初始化。计数器最好使用通用寄存器，也可使用内存单元。计数方式可采用顺计数器也可采用倒计数器。

[例 4．8] 求斐波那契序列的第 20 项 F20

斐波那契序列 F1，F2，…是这样定义的：

第 1 项 F1=0，第 2 项 F2=1，以后各项都是它前面两项之和，于是，这个序列为：

0，1，1，2，3，5，8，13，…

这个著名的序列是 Fibonacci 在 1202 年首先提出来的。

本例根据 Fibonacci 序列的定义，我们很容易得到下面的递推关系：

F1=0
F2=1
F_{i+2}=Fi+F_{i+1} i≥1

根据这个递推关系，从 F1 及 F2 出发，每次求出一个新的斐波那契数，即以后各项是这前面两项之和。如此操作，历经 18 次即可求出 F20.其操作流程如图 4．20 所示。

在该流程中，引入两个工作单元，分别记 Fi 和 F_{i+1}，由于 Fi＋F_{i+1}→F_{i+2}。，当得到新的一项后，第 1 项 Fi 就不必保存，因此可用 FI 保存第 1 项，称为旧值,F 保存第 2 项，称为新值。由 FI
及 F 求出新值时送人 FI，冲去第 1 项旧值，接下去可准备新的一次运算。但是仍由

FI＋F→FI

进行新的一次运算，冲去的是新值而不是旧值。因此，有必要调整循环体，使得循环结构简洁而又是正确的。事实上，只需在每 次求出新值冲去旧值 FI 时，将 FI 和 F 值对换，使得 FI 始终保
存的是旧值,F 保存的是新的当前值。

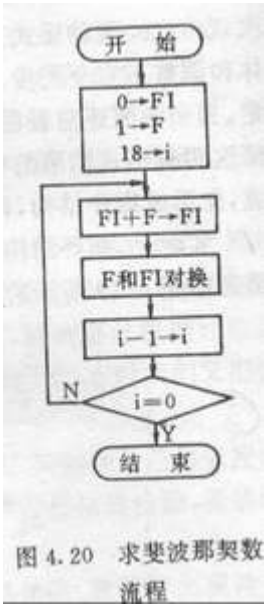


图 4.20 求斐波那契数流程

；程序 4．8：求 F20

```
        START
        LEA     GR0,0          ;GR0 作为 FI,初值为 0
        LEA     GR1,1
        ST      GR1,F          ;F;初值为 1
        LD      GR3,N
        LEA     GR3,-2,GR3     ;计算器 GR3 的初态
LOOP    ADD     GR0,F          ;新的斐波那契数在 GR0 中
        LD      GR1,F          ;GR0 和 F 内容对换
        ST      GR0,F
        LEA     GR0,0,GR1      ;
```

```
        LEA     GR3,-1,GR3  ;计数
        JNZ     LOOP        ;未完,续
        EXIT
F       DS      1
N       DC      20
        END
```

程序中选择 GR0 替代 FI，工作单元 F 存放当前的新值，当程序运行结束时 F 中保存的即为 F20。

在编制循环程序中，计数控制的方法往往采用倒件计数器，本例的计数器 GR3，初态是 18，循环体每执行 1 次，计数器减 1，当计数器递减到其值为零时，终止循环。采用倒计数器方法的好处是可读性好，因为在执行循环体之前循环的次数已明显地在预置部分给出。

[例 4.9] 有一万名考生某科的考试成绩在内存 SAMPL 开始的连续单元中，要求编程统计出各种得分（百分制）的人数，并把统计结果存储在 STAT 开始的连续单元中（限用一重循环）。

解题思想：本例按照一般的思路是，要统计 0～100 分中各个得分的人数，先从 0 分开始，看全部考生中，得 0 分的有几人；接下去，再看全部考生中，得 1 分的有几人；……最后一遍是统计全部考生中得 100 分的人数。这样经过 101 个遍历，且每个遍历是查询 1 万名考生的分数，即有 10000×101 次的计数操作才完成这个任务。按照这个思路，必须编制一个二重循环程序，方才能达到目的。

采用跳跃表计数，即散加的方法，可以把这问题简化为一重循环的结构。按照这个新的思想，直接给出程序，并请读者完善这个程序。

；程序 4．9：用散加统计各种得分的人数

```
        START
        LEA     GR0,0           ;将 START 缓冲区冲 0
        LEA     GR2,101
CLEAR   LEA     GR2,-1,GR2
        ST      GR0,STAT,GR2
        JNZ     CLEAR



LOOP    LD      GR1,SAMPL,GR2   ;取分数
        [          ]
        [          ]            ;散加计数
        [          ]
        LEA     GR2,1,GR2       ;指向下一考生
        CPA     GR2,C10000      ;完了吗?
```

```
        JMI     LOOP        ;未完,续
        EXIT                ;完
C10000  DC      10000       ;比较常数
STAT    DS      101         ;统计区域
SAMPL   DS      10000       ;考生分数区域
        END
```

这个程序采用顺计数方法，计数器 GR2 初态为 0，每查询一个考生成绩，做（GR2）＋1=＞GR2，直（GR2）=10000 结束统计。同时 GR2 兼作为考生分数的指针，即查询分数时，GR2 作为变址器，由

E=（GR2）十 SAMPL

形成每个考生分数的地址，这种方法在表格数据处理中，不失为是一个好的方法。

[例 4．10] 求两个无符号数的乘积。

解题思想：COMET 计算机上无符号数就是一个 16 位二进制的代码，全部作为数值，没有符号的处理问题。两个无符号数相乘，其积最多可达 32 位，需要两个单元存放。这里带来两个问题需要解决。一是低位部分积相加时的进位为部分积高位的一部分，不能丢 失；二是部分积的高 16 位与低 16 位的数据传递问题要保持连贯。

从一个具体例子，导出无符号整数相乘的算法。

假设被乘数

$p=118_{(10)}=01\ 110\ 110_{(2)}$

乘数

$q=11_{(10)}=00\ 001\ 011_{(2)}$

p×q 手算过程是

```
   01 110 110  ……被乘数   部分积
×  00 001 011  …… 乘数    00 000 000(初始)
   01 110 110                01 110 110 (第一次)
   011 101 10               101 100 010 (第二次)
 0 000 000 0               0 101 100 010 (第三次)
01 110 110                 10 100 010 010 (第四次)
 10 100 010 010 ……积
```

其方法是自右至左考察乘数的各位，根据乘数相应位是 1,则将被乘数加到部分积上。若乘数的相应位是 0，则加上全 0，然后把部分积右移一位（它的作用相当于手算时，被乘数左移 1 位相加）。对于 16 位二进制正整数的乘法，只要如此重复 16 次即可得到乘积。图 4．21 描述了求两无符号整数乘积的流程。

图 4．21 的流程仅给出思路，如何用 CASL 汇编语言实现，还需要解决下面两个问题。因为积是 32 位，不妨设高位积放在 MH，低位积放在 ML 中，这样，被乘数和 MH 相加形成的部分积，逐次右移而将低位送入 ML。

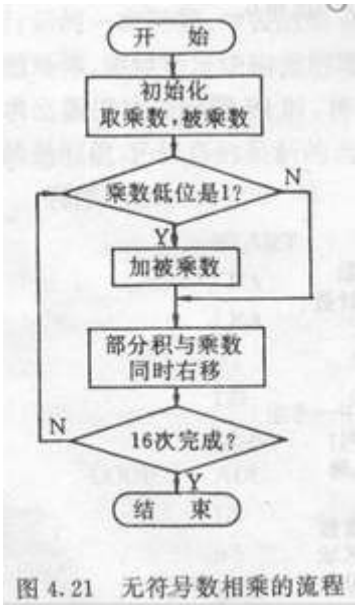


图 4.21 无符号数相乘的流程

（1）（MH）＋（P）=>MH 最高位进位是部分积的 组成部分，必须保存起来，以便部分积右移作为积的数值组成部分。

（2）低位积 ML 是通过 MH 右移如何实现:

$(MH)_{15} \rightarrow (ML)_0$

即 MH 的最低位移入到 ML 的最高位中。

由于 COMET 计算机没有进位的设置，使得上述两种操作变得困难起来。因为两个单元或寄存器之间信息传递没有桥梁可用，只能用以下似乎笨拙方法实现，但这恰恰给我们启迪，CASL 汇编语言中如何实现双倍位运算，以下不失为一条解决问题的途径。

为了使（MH）＋（P）的最高位进位不丢失，我们可将计算（MH）＋（P）分成两部分，即将 16 位求和分成第 0 位及第 1～15 位分别求和。注意到求和后的部分积需右移 1 位,则第 0 位的求和正好是加到移位后的 MH 的第 1 位上，而最高位的进位自然就加到 MH 的第 0 位上。现在只需截下（MH）0，（P）0 右移 1 位再相加即可。

第 2 个问题是如何实现 MH 向 ML 双字长移位。只需截下$(MH)_{15}$,左移 15 位，即对应 ML 的第 0 位的位置，只要在 ML 右移 1 位之后，相加即可得到双字长移位的实际效果。读者可从阅读程序 4．10 过程中，体会实现两无符号数相乘而得到双倍位积的方法。

;例题 4．10 两无符号整数相乘的程序:

```
        START
        LD      GR0,C8000   ;#8000
        AND     GR0,P
        SRL     GR0,1
        ST      GR0,P0      ;(P)0 1→ =>P0
        LEA     GR0,32767   ;#7FFF
        AND     GR0,P
        ST      GR0,P1      ;(P)1~15=>P1
        LEA     GR2,0
        ST      GR2,MH      ;0=>MH
        ST      GR2,ML      ;0=>ML
        LEA     GR1,16      ;16→I
LOOP    LEA     GR0,1
        AND     GR0,Q
        JZE     NADD        ;(Q)15=0 转 NADD
        LD      GR0,C8000
        AND     GR0,MH
        SRL     GR0,1
        ADD     GR0,P0      ;(MH)0+(P)0=>GR0
        LEA     GR2,32767
        AND     GR2,MH
        ADD     GR2,P1
        ST      GR2,MH      ;(MH)1~15+(P)1~15=>MH
NADD    LD      GR2,Q
        SRL     GR2,1
        ST      GR2,Q       ;(Q) 1→ =>Q
        LD      GR2,ML
        SRL     GR2,1
        ST      GR2,ML      ;(ML) 1→ =>ML
        LEA     GR2,1
        AND     GR2,MH
        SLL     GR2,15
        LR      GR2,ML
        ST      GR2,ML      ;(MH)15*2^(15+(ML)=>ML
        LD      GR2,MH
        SRL     GR2,1
        ST      GR2,MH      ;(MH) 1→ =>MH
        ADD     GR0,MH
```

```
        ST      GR0,MH      ;(GR0)+(MH)=>MH
        LEA     GR1,-1,GR1  ;16 次完?
        JNZ     LOOP        ;未完,转 LOOP
        EXIT
P       DS      1           ;放被乘数
Q       DS      1           ;放乘数
MH      DS      1           ;积的高位
ML      DS      1           ;积的低位
P0      DS      1
P1      DS      1
C8000   DC      #8000
        END
```

[例 4．11] 编制程序，其功能是从输入设备上（键盘）输入 1—4 位的十六进制数，并将其转换成二进制数存放于 GR1。

若输入为空记录，GR1 中置零，程序结束。若输入的数字串超过 4 位，或输入十六进制数以外的字符，报输入出错并重新输入。

解题思想：将十六进制数转换成二进制数主要解决两个问题：

(1)从输入设备上输人的是十六进制的数字串，数字字符为 0—9，A—F 的 ASCll 代码。因此，一要判定每个输入字符是否为十六进制数字范围，二要把每一个十六进制数字，转换成它对应的二进制值，即 0—9 减去 0 的 ASCll 码值，A～F 减去 A 的 ASCll 码值再加 10 ．

(2)转换的方法是按如下项展开,从高位到低位，逐个进行乘法运算，拼成二进制数

$B=h0\times16^3+h1\times16^2+h2\times16+h3$

$=((h0\times2^4+h1)\times2^4+h2)\times2^4+h3$

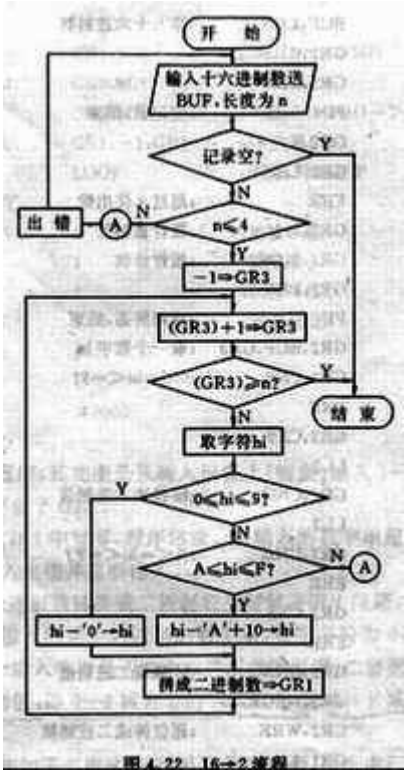
其中，hi（i=0，1，2，3）是十六进制数字对应的二进制值，乘上 2^4 用左移 4 位实现。

程序 4．11 是 WHILIJ 模式的循环结构，先判定后处理。程序中寄存器的使用分配是：GR1 存放结果为最终转换的二进制数，如果记录为空，其内容为零。GR2 存放十六进制数字 hi 及转换的数字二进制值。GR3 变址器，作为数字串缓冲区 BUF 的指针,初态置一 1 是因为先修改后判定,第 1 个数字 h0 的地址是指针(GR3)=0 的地址

BUF＋（GR3）

这里 GR3 兼作计数器，当 GR3 内容为 0-n-1 时应取字符 hi，当（GR3）≥n 时，无字符应结束。作为计数控制的初值 n 是在输入记录之后由宏指令 IN 执行后填入 LEN 中，循环体执行之前，LEN 的内容是已知的，因此这个程序也是计数控制的循环。

;程序 4.11: 16→2




```
        START
GET     IN      BUF,LEN     ;输入十六进制数
        LEA     GR1,0
        CPA     GR1,LEN
        JPZ     FIN         ;空记录,结束
        LEA     GR2,4
        CPA     GR2,LEN
        JMI     ERR         ;超过 4 位出错
        LEA     GR3,-1      ;指针置初值
LL1     LEA     GR3,1,GR3   ;指针修改
        CPA     GR3,LEN
        JPZ     FIN         ;指向终态,结束
        LD      GR2,BUF,GR3 ;取一个数字 hi
        CPA     GR3,CN0     ;0<=hi<=9?
        JMI     ERR
        CPA     GR3,CX9
        JPZ     LL2
```

```
        SUB     GR2,CN0     ;hi 换成二进制值
        JMP     LL3
LL2     CPA     GR2,CNA     ;A<=hi<=F?
        JMI     ERR
        CPA     GR3,CXF
        JPZ     ERR
        SUB     GR2,CNA     ;hi 换成二进制值
        LEA     GR2,10,GR2
LL3     ST      GR2,WRK     ;逐位拼成二进制数
        SLL     GR1,4
        ADD     GR1,WRK
        JMP     LL1
ERR     OUT     MES,L9      ;出错处理
        JMP     GET
FIN     EXIT
CN0     DC      '0'
CX9     DC      58          ;'9'+1
CNA     DC      'A'
CXF     DC      71          ;'F'+1
WRK     DS      1
BUF     DS      80
LEN     DS      1
MES     DC      'INPUT ERR'
L9      DC      9
        END
```

[例 4．12] 编制程序，其功能是把寄存器 GR1 中的二进制代码转换成 4 位十六过制数在输出设备上（显示器）输出。

解题思想：本例的 2→16 为[例 4．11]的逆向转换。一个 16 位的二进制代码正好对应是四位十六进制数，包括高位的 0，只要依次截下 4 位二进位代码，将它变成十六进制数字字符的 ASCll 码即可输出。注意到十六进制数用 0—9，A—F 来表达的。因此截下的 4 位二进位代码在 0—9 范围，则加‘0’（数字 0 的 ASCll 码），否再加 7 即可（因为 9 之后还间隔 7 个字符才是字符 A）4 位二进代码只可能是在 0—9 范围或 10—15 范围，非此及彼。

其次，十六进制数字串的输出采用宏指令 OUT，要待 4 位十六进制数字串全部存入缓冲 BUFFER 之后，执行 OUT 宏指令一齐输出。故转换的顺序是先低位后高位存储，而输出时是先高位后低位，正好后进先出。图 4．23 是描述这个算法的流程。

图 4．23 的 2→16 所描述的是把（GR1）=b 的二进制代码转换成十六进制数字串 h0h1h2h3 的一种算法，从低位 h3 开始到高位 h0，依次存入 BUFFER 会，其中

i 用 GR2 作变址器，因采用的是先进后出的输出次序，故 GR2 初态用 4，程序中采用先修改后判定的结构形式，所以首先是按（GR2）=3 作指针，存入地址是

BUFFER＋（GR2）

本例也是 WHILE 模式的循环结构,兼用 GRZ 作计数器。固定输出 4 位，循环次数是 4，即

;程序 4.12: 2→16

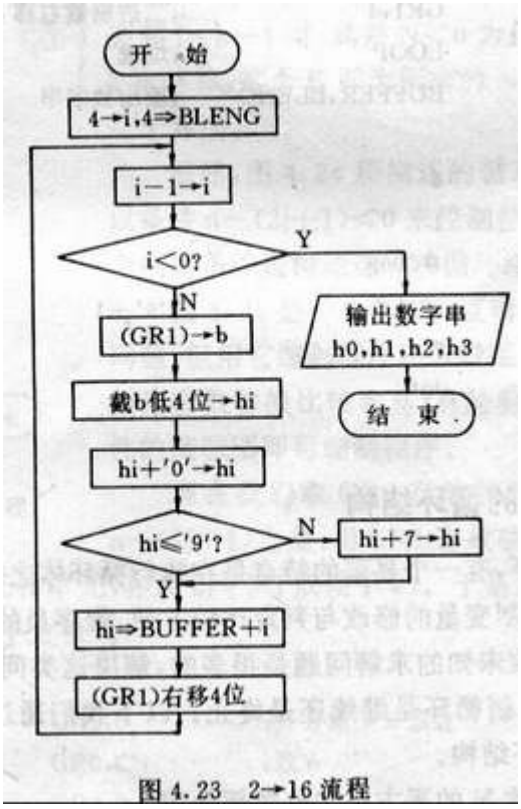


图 4.23 2→16 流程

```
        START
        LEA     GR2,4           ;变址器初态
        ST      GR2,BLENG       ;准备输出 4 个字符
        LEA     GR2,-1,GR2      ;变址器修改
        JMI     LAST            ;<0,结束
LOOP    LEA     GR0,0,GR1       ;取二进制数
        AND     GR0,CF          ;截 4 位二进位
        ADD     GR0,C0          ;变成数字的 ASCII 码
        CPA     GR0,C9          ;超过'9'?
        JMI     L1              ;不超过
        ADD     GR0,C07         ;超过'9',是 A-F
L1      ST      GR0,BUFFER,GR2  ;自低位到高位存入数字
        SRL     GR1,4           ;二进制数右移 4 位
        JMP     LOOP            ;继续
```

```
LAST    OUT     BUFFER,BLENG   ;输出数字串
        EXIT
BUFFER  DS      4
BLENG   DS      1
CF      DC      #000F
C9      DC      58             ;'9'+1
C07     DC      7
C0      DC      '0'
        END
```

4·3·3 条件控制的循环结构

前述的计数控制循环，有一个显著的特点是在执行循环体之前，重复执行的次数是已知的。这类循环程序对控制变量的修改与判定比较方便，程序员的注意力可集中在循环体的实现上。然而循环次数未知的求解问题是很多的，解决这类问题是要设法找出某种条件，用条件满足与否来控制循环是继续还是终止。以下我们通过实例说明如何选择“条件”，建立条件控制的循环结构。

[例 4．13] 求正整数 N 的平方根之整数部分。

解杨思想：本例只考虑 N≥0 的整数，负数不予讨论。

怎样来考虑求 N 的平方根问题？考察和式

S=1+3+5+…+(2n+1)

根据等差数列的求和公式有

$S=((a1+an)/2)*n=(((1+(2n-1)))/2)*n=n^2$

从而对任何正整数 N 总有

$N=n^2+\varepsilon$

其中 0≤ ε <2n+1，即：

$$
\begin{aligned}
N&=1+3+5+\cdots+(2n+1)+\varepsilon\\
&=(2*0+1)+(2*1+1)+\cdots+(2*(n-1)+1)\ \varepsilon\\
&=\Sigma\,(2*(j-1)+1)+\varepsilon\quad (j=1,2,\cdots,n)
\end{aligned}
$$

显然求和的项数 n 即为 N 的平方根的整数部分。

从等差级数求和公式出发，依次从 N 中减去 1，3，…，直到 2n-1 之后就不能再减了.

因为再减 2n＋1，差小于零。虽然不能予先知道 n 的值，但只要相减的差小于零即可结束这个重复的操作过程，此时减去等差级数的项数 n 即为 N 平方根的整数部分。

为了方便编制程序，现把求和公式改写为

$N=\Sigma(2*j+1)+\varepsilon$ (j=1,2,…,k)

j 从 0 出发，逐次运算

N-(2j+1)→N

直到 j=k-1 时，其差 N<0 为止。j 从 0 到 k—1 共执行 k 次运算，这个 K 即为所求的 N 平方根。图 4．24 表达了这个算法。

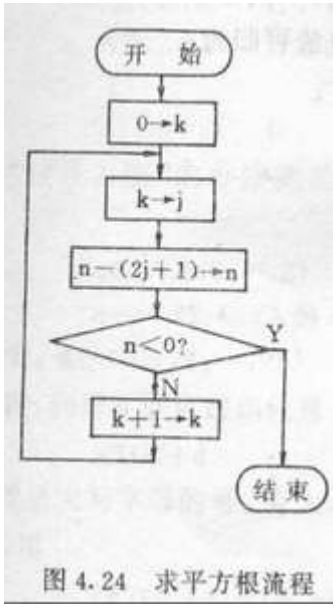


图 4.24 求平方根流程

显然，图 4．24 所描述的循环是循环次数未知，它是以条件 n-(2j+1)＜0 来控制循环的终止。可见，选择这个判定条件是构造这一种循环结构的关键。

图 4．24 是一个算法的逻辑流程，它易于理解求解的问题，但用它编制程序还有段距离。有时问题比较简单，循环参数替换比较容易，有经验的程序员也可不画工程性的流程图即可编制程序。

现在我们取 GR0 存放被开方的数 n，并依次存放 n-(2j+1)之差；用 GR1 存放项 k，也是终止循环时的开方结果；用 GR3 作中间工作单元，存放 2j+1(j 依赖 k)。于是，得到程序 4．13

；程序 4．13：求平方根

START

```
        LEA     GR1,0           ;开方结果 预置 0
        LD      GR0,CN          ;取 n
LOOP    LEA     GR3,0,GR1       ;k →j
        SLA     GR3,1
        LEA     GR3,1,GR3       ;形成 2j+1
        ST      GR3,WORK
        SUB     GR0,WORK        ;n-(2j+1)k →jn
        JMI     EXT             ;n<0 结束
        LEA     GR1,1,GR1       ;k+1 →k
        JMP     LOOP            ;继续
EXT     EXIT
CN      DC      n               ;存放开方数
WORK    DS      1
        END
```

[例 4·14] 编制程序,求给定的两个正整数 A 和 B 的最大公约数,结果送到 GCD 单元中。

解题思想：利用大家熟知的欧几里得辗转相除法求最大公约数。其算法如下：

（1）假定两个数 A,B 中较大者是 A 较小者是 B。

（2）求 A 除以 B 的余数是 R1。

（3）求 B 除以 R1 的余数是 R2。

（4）求 R1 除以 R2 的余数 R3。

（5）以下依次类推，用新得到余数除前一个余数，求出下一个余数。

（6）经过若干次除法后，余数变为 0，此时的除数便是 A 和 B 的最大公约数

程序的实现：为了构成一个条件循环结构的程序，上述算法可归为：

①A÷B 余数 R，除法可用减法实现

②R=0 则 B 即为所求，否则 B→A，R→B；转①

；例 4．14 程序，求最大公约数

```
        START
        LD      GR1,A           ;a≥b?
        CPA     GR1,B
        JPZ     X1              ;是的,转 X1
        LD      GR2,B           ;否则 A,B 交换
        ST      GR1,B
```

```
        ST      GR2,A
X1      LD      GR2,B
        ST      GR1,GCD     ;预送最大公约数
        LD      GR1,A
X2      SUB     GR1,B       ;a-b=>GR1
        JPZ     X2          ;≥0,继续
        ADD     GR1,B       ;不够减,恢复余数 R
        JZE     X3          ;R=0,GRD 中已为最大公
                            约数
        ST      GR1,B       ;R→B
        ST      GR2,A       ;B→A
        JMP     X1          ;下一轮辗转相除
X3      EXIT
A       DC      a
B       DC      b
GCD     DS      1           ;结果为最大公约数
        END
```

[例 4·15] 在内存缓冲区 TEXT 中，存放不超过 160 个字符的文本，每个单元自右至左存放两个字符的 ASC11 码，文本是以“！”符号结尾。试编制程序，统计 TEXT 缓冲区中英文大写字母出现的频度。

解题思想：一个字符的频度是指它重复出现的次数。按一般而言，统计一个字符出现的次数，是要扫描整个文本，方知它的个数，这显然是一个循环结构的程序。统计 26 个大写字母各自的频度则要把文本扫描 26 遍，按题意要求，这需要编制一个二重循环.

现在我们采用跳跃表（散加）方法，只需一重循环就可解决这个问题。以下我们从三方面来回答这个问题：

①跳跃表计数。从散转那一节已经知道，对于特定的多分支结构，散转是个高效的处理方法。跳跃表方法是散转结构中另一种特定形式，它根据控制字对应单元，实行散转操作。本例的散转控制字是这样确定的，因为大写字母 A—Z 作为字符是以 ASC11 码存在的，而且它们是从 65 到 90 的顺序（十六进制是 41—5A）依次排列。用 26 个连续单元（假定表头是 ATOZ 人对应大写 26 个字母，如下形式：

```
ATOZ+0      计 A 的个数
ATOZ+1      计 B 的个数
…
ATOZ+25     计 Z 的个数
```

用“对号入座”的办法找到任意一个大写字母所在的位置。如何确定各字母的“号”？

令

б ∈(A,B,C,…，Z）

d=б-65（A 的 ASCll 码值为 65）

则有：d∈（0，1，2，…，25）

这样，利用 d 就可找到任意一个大写字母对应的地址是

ATOZ＋d

d 就是大写字母的号。有大写字母就有对应 d，有控制字 d，就可找到该字母对应的元，用

（ATOZ＋d）＋1=＞ATOZ＋d 5

的跳跃表计数方法，只需一次扫描，就可完成对全文本的所有大写字母频度的计算

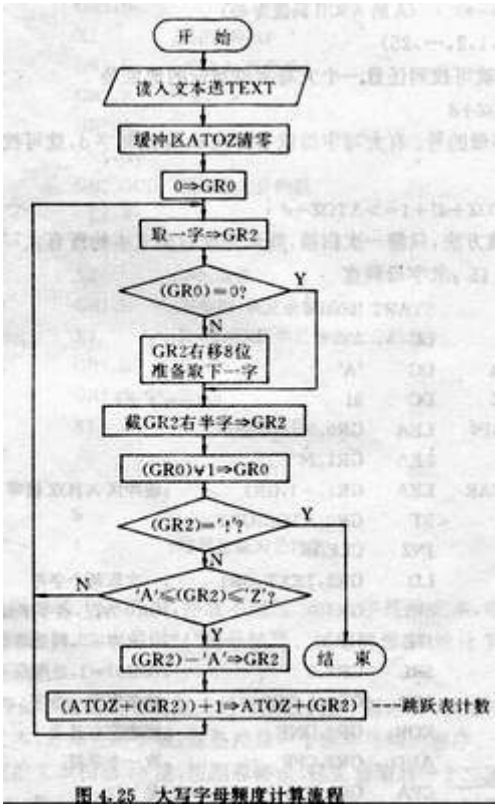


图 4.25 大写字母频度计算流程

；程序 4．15：求字母频度

```
        START
        BEGIN
C33     DC      33          ;'!'
CHA     DC      'A'
```

```
CHZ     DC      91              ;'Z'+1
BEGIN   LEA     GR0,0
        LEA     GR1,26
CLEAR   LEA     GR1,-1,GR1      ;缓冲区 ATOZ 清零
        ST      GR0,ATOZ,GR1
        JNZ     CLEAR
L0      LD      GR2,TEXT,GR1;一次取两个字符
        SRL     GR0,0           ;GR0 为左,右半字处理标
                                志器
        JZE     L1              ;(GR0)=0,转处理右半字
        SRL     GR2,8           ;(GR0)=1,处理左半字
        LEA     GR1,1,GR1       ;准备取下一个字
L1      EOR     GR0,ONE         ;标志左右开关
        AND     GR2,CFF         ;取一个字符
        CPA     GR2,C33         ;是'!'?
        JNZ     L2              ;不是,转处理
        EXIT
L2      CPA     GR2,CHZ         ;判别为大写字母?
        JPZ     L0
        SUB     GR2,CHA
        JMI     L0
        LD      GR3,ATOZ,GR2;每个大写字母分别计数
        LEA     GR3,1,GR3
        ST      GR3,ATOZ,GR2
        JMP     L0              ;继续处理
ATOZ    DS      26              ;大写字母计数缓冲区
TEXT    DS      80              ;文本缓冲区
CFF     DC      #FF
ONE     DC      1
        END
```

②取字符处理。一个字符的 ASC11 码占一个字节（半个字），文本缓冲区中存储的是一个单元存 2 个字符，而 COMET 计算机指令访问内存是以字为单位，即一次访问(例如取数),是 2 个字符，一次处理右半字，另一次处理左半字。左、右处理有先有后，因此要设置左右控制标志器，这样把字操作转换为节操作，且有条不紊。选取 GR0 为左右控制标志器当

(GR0) = 0 处理右字节
(GR0) = 1 处理左字节

并且指令 EOR GR0，ONE 担任左、右交替的开关操作。

③控制循环的条件。文本的长度是未知的。因此不能用计数控制。题意已明显告诉我们，文本以“！”结尾，“！”就是一个控制条件，因为‘！’=33，每取一个字符与 33 比较，不相同则循环继续，相同的终止循环。

题意中约定文本是预先存储在 TEXT 缓冲区中。在读完程序 4．15 之后，试问读者，为什么 ATOZ 缓冲区要预先清零？

[例 4．16] 编制程序，其功能是在键表 HEAD 中找最大元送 MAX，其大元地址送 PTR。设线性链表 HEAD 中有 n 个元素，且各元素由两个相邻单元组成，形如：

<table>
<tr><td rowspan="2">ADDRESS</td><td>DATA</td></tr>
<tr><td>LINK</td></tr>
</table>

即第 1 个字是数据 DATA，第 2 个字是下一元素的地址信息 LINK，即链接指针。当

i≠j 时，Ai≠Aj 且 Ai≠0 i,j∈（1，2，…，n）

其中第 1 个元素地址(例如 Ai)存放在 HEAD 单元中，末元素(例如 An)的 LINK=0,即（An＋1）=0。

解题思想：勾链存储线性表中各元素之间，不象顺序存储的线性表中各元素之间是等距离的，本例勾链表元素虽由二个单元组成，但它（如 Ai）和它的后继元素地址之间一般情况是 Ai＋2≠A_{i+1} i=1，2，…，n-1

这说明表中元素的存储位置是任意的（当然要满足某种规律的约制）。因此，处理这种“离散”情况，不能象处理顺序表那样，把 n 作为已知个数，地址修改按等步长增减。一般地说，要访问勾链表中某元素的数据 ai，则要知道它的地址 Ai，而要知道 Ai，又必须知道它的前驱元素的地址 A_{i-1}，以此类推，直到找到第一个元素地址 A1，正好 A1=（HEAD）。由此可知，访问勾链表中的元素，需从勾链表指针 HEAD 所指示的第一个元素地址出发，沿着各元素的地址信息，得到它的后继元素地址，逐个访问，能遍历整个勾链表。由勾链表的末元素 An 的地址信息 LINK=0，即（An＋1）=0，说明在 An 之后，不再有新的后继元素了。因此它可用来作为控制循环结束的条件。

求最大值的方法是表中每一个元素和第一个元素比较一遍，每作一次比较，保存大元及它的地址，一遍扫视完了之后，最大值就找到了。因为数据可能有负数，在处理时，先送一个负数-2^{15}在 MAX 中，用 MAX 去和每一个元素比较，大者留在 MAX 中。

本例我们直接给出程序 4．16，请读者自己画出流程图，加深对处理勾链关系这一类问题的理解。

;程序 4.16:求勾链表中极大值

```
        START
        LD      GR0,C8000     ;预置 MAX 中最小的数
        ST      GR0,MAX       ;取第一元素地址=>GR1
        LD      GR1,HEAD      ;取数据 ai =>GR0
LOOP    LD      GR0,0,GR1     ;ai 是在的?
        CPA     GR0,MAX       ;否,准备取下一元素的地
                              址信息
        JMI     LINK          ;是大的送 MAX
        ST      GR0,MAX       ;大元地址送 PTR
        ST      GR1,PTR       ;取下一元素地址送 GR1
LINK    LD      GR1,1,GR1     ;末元素到了?
        LEA     GR1,0,GR1     ;未到,继续
        JNZ     LOOP
        EXIT
HEAD    DS      1
MAX     DS      1
PTR     DS      1
C8000   DC      #8000
        END
```

4.5 例 题

前述交换排序和冒泡排序，一个共同特征是不断地交换两个元素，通过移动表元素，例如将小的元素调到缓冲区之前，大的元素调到后面，使得表元素成为递增序列。在表元素分类算法中，有许多都是对元素进行大量移动，如果表很长，则花费时间很多。下面给出两个不移表分类方法。

[例 4. 20] 已知数表 TBL=（a1，a2，…，an），要求编制程序生成一个新数表 ADR=（b1，b2，…，bn），其 bi(i=1,2,…，n)是数据表 TBL 中对应元素 ai 在表中的大小编号。约定大的元素编号小，最大的元素编号为 1，小的元素编号大，相等的元素编号相同。

解题思想：本例要编制的程序是能够对约定表元素按大小顺序进行编号的程序。这种编号分类是不打乱表元素在原来缓冲区中原存储位置，保持数据的初始顺序。

对表 TBL 中元素按大小编号是建立一个对照表 ADR，由表 ADR 中元素 bi，即可知对应的表元素 ai 在 TBL 表中的地位。这种以元素大小次序进行编号，是表元素分类的一种形式。在某种情况下，可以通过表 ADR 中各元素 bi 作为索引，将 TBL 表中各元素传送到一个表长也为 n 的新表 SORT 中，则在 SORT 表中就是 TBL 表中各元素经过分类的结果序列。

为了确定 TBL 表中某元素在全部元素中的编号，该元素要与所在其他的元素进行比较。比较的方法是类似选大元方法，即首先固定 a1，自 a2 到 an 依次与 a1 进行比较；第二遍是固定 a2 自 a3an 依次与 a2 进行比较；依次类推，直到最后一遍是 a_{n-1} 与 an 进行比较。这样经过 n 一 1 个遍历的比较，TBL 表中全部元素经过两两组合的比较，每个元素的大小编号就确定了。不过，在选大元方法中，是大的元素和固定的那个元素对换，现在的方法是元素不动，确定其编号。

如何在两两比较过程中，记下各自的大小编号呢？对任意的 ai 与 aj，如下的方法可记下它们的大小编号：

（1）预置表 ADR 中各元素 bi=1

（2）固定 ai(i=1,2,…n-1)

当 ai>aj 则 bj+1→bj (j=i+1,…n)

当 ai<aj 则 bi+1→bi

当 ai=aj 则 bi,bj 不变

图 4.31 就是这个算法的程序流程,程序见程序 4.20.

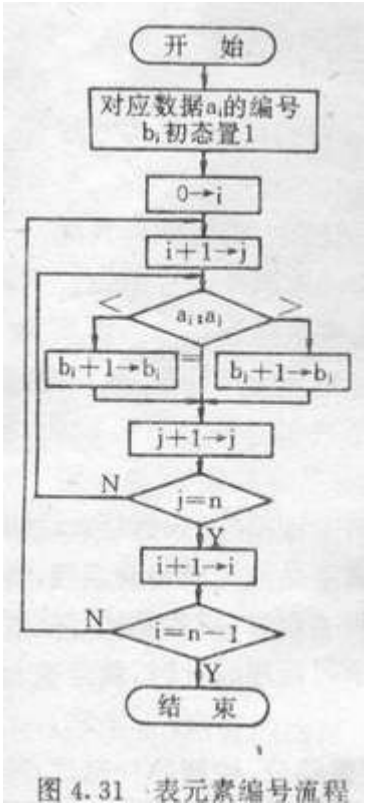


图 4.31 表元素编号流程

;程序 4.20:表元素编号程序

```
START PRGRM
ONE         DC      1
CN          DC      N
TBL         DS      N           ;数据表空间
ADR         DS      N           ;数据对应的大小编号
                                (1-N)
;程序开始时将 ADR 空间预置 1
PRGRM       LD      GR1,CN
            LEA     GR0,1
P1          LEA     GR1,-1,GR1
            ST      GR0,ADR,GR1
            JNZ     P1
P2          LEA     GR2,1,GR1   ;i+1→j(初态 i=0)
;ai 与每个 aj 比较,小者其编号增 1
P3          LD      GR0,TBL,GR1 ;取 ai
            LEA     GR3,0,GR1   ;当 ai<aj,预置 j
            CPA     GR0,TBL,GR2 ;ai:aj
            JZE     P5          ;ai=aj 转
            JMI     P4          ;ai<aj 转
            LEA     GR3,0,GR2   ;当 ai>aj,预置 j
P4          LD      GR0,ADR,GR3 ;bi+1→bi,或
                                bj+1→bj
            ADD     GR0,ONE
            ST      GR0,ADR,GR3
P5          LEA     GR2,1,GR2   ;j+1→j
            CPA     GR2,CN
            JNZ     P3          ;ai 继续与新的 aj 比较
            LEA     GR1,1,GR1   ;i+1→i
            LEA     GR2,1,GR1
            CPA     GR2,CN
            JMI     P2
            EXIT
            END
```

表插入排序是二个速度快的分类法，它的基本思想是，把一个元素 ai 插入到一列有序元素 a1，a2，…，an 中，使长度为 i 的结果序列也是有序的。它的方法是：

①虚设一个元素 a0，存放要插入的元素 x（x=a2，a3，…，an）。

②在已排序 a1，a2，…，an 中找到要插入元素 x 的位置，在与 ai-a_{i-1}进行比较时，把比 X 大的元素向后移动 1 个位置。

③把 X 元素插入到相应位置。

表插入法是对表元素 a1，a2，…，an 自 a1 出发逐次插入 a2，a3，…，an，它也是要移动表元素，只是在 ai 元素大于 a_{i-1} 时提前结束一个遍历中的比较及移表。这算法的程序在第 6 章中给出，但它还需要读者去完善它。

下面给出一个勾链式的表插入分类算法，它不必移动表元素，在排序过程中，元素的位置要变化，不是移动元素本身，而是修改元素的指针，使它反映表中各元素次序在分类时的变化情况。分类结束时，各元素的排序次序是用链表连接起来的。

不必移表可以大大节省运算时间，但要以增加存储开销为代价，即一个元素除本身的数值部分外，还需要增加一个指针部分，以记载下一元素的地址信息。

[例 4．21] 假定数表 A 中有 n 个元素 a1，a2，…，an 每个元素 ai 由两个字组成，第 1 个字是元素的数值部分，第 2 个字用作指针。现要求采用勾链式表插入分类将表 A 中元素按数值的上升顺序进行排序。即排序过程不移动表元素，其结果序列是以元素的一个指针链来表示表元素的递增次序。

解题思想:为了方便描述勾链式表插入法的程序结构,先给出数表 A 的存贮结构，虚设一个元素 a0，假定有缓冲区，自 A＋2 开始依次有 2n 个单元存放待排序的元素 a1，a2，…，an,其中每个元素的数值部分存放在 A＋2，A＋4，…，A＋2n，排序别后各元素仍留在原来位置；而 A＋1，A＋3，…，A＋2n＋1 的内容体现了各元素间的次序。约定 A 十 1 为排好序的表头，不妨命名为 AT0，即（A ＋1）之值是排好序的序列中第一个元素的序号，（A＋2i＋1）之值是第 i 个元素的下一个元素的序号。一般情况

（A＋2i＋1）≠i＋1

例如，待排序的一列数值是：28，13，25，72，85，39，6，41，则排序完成后，其存储结构如表 4.1.

表 4.1 按勾链式排好序的内存结构表

序号 i	地址	数值 ANi	指针 ATi
0	A	(空)	7
1	A+2	28	6
2	A+4	13	3
3	A+6	25	1
4	A+8	72	5
5	A+10	85	0
6	A+12	39	8
7	A+14	6	2
8	A+16	41	4

勾链式的表插入和移表式的表插入都是自元素 a1 出发，逐次插入 a1，a2，…，an，只是勾链式排序在程序执行过程中，动态地修改表头，以及不断地修正当前元素、前驱元素、后
继元素的指针值。图 4．32 是反映这个算法的流程。

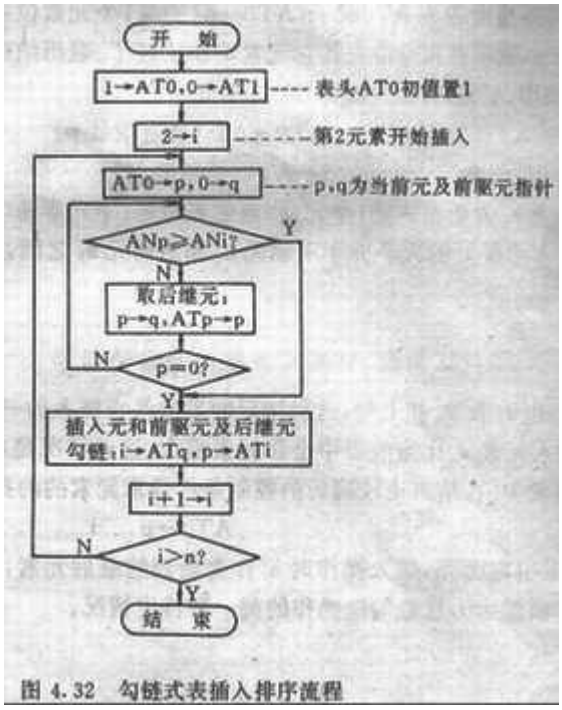


图 4.32 勾链式表插入排序流程

根据表插入分类法规则，是把元素 ai 插入到已排好序的表元素 a1，a2，…，a_{i-1} 中（称此表为当前表），使长度为 i 的结果序列也是有序的。为能找到 a；插入的适当位置，用 ai 和当前表各元素 ap（p=1，2，…，i-1）进行比较。不妨假定用指针 i 指示插入元素 ai，用指针 p 指示被比较的当前元素 ap，指针 q 表示当前元素的前驱元素 aq，由于当前表是单向有序的链表，因此对当前表作遍历比较要从表头 AT0（q=0）开始，依各元素的指针所指示的下一元素 ap 逐一比较。元素之间的大小比较，是以元素的数值部分 ANi 与 ANp 比大小，当

ANi＞ANp

时，要再和 ap 的下一个元素作比较。于是当前元 ap 作为前驱元，后继无递升为当前元，故指针调整为

p→q

ATp→p

直到有 ANi≤ANp 成立时，找到 ai 的插入位置是在 aq 与 ap 之间，当前元 ai 的后继元。

勾链式的表插入不作表元素的移动，只要调整指针信息：

①插入元 ap 的位置信息 i，更新前驱元的指针值，有

i→ATq

②后继元 ap 的位置 p，填入当前元 ai 的指针部分，有

p→ATi

则完成 ai 元素插入当前表的任务，且使表长为 i 的新表仍为递增有序的。这个过程见图 4．33。

表插入的三种情况：

（1）插入元素 ai 比当前表的元素都小,这种情况是在进入比较遍历的第 1 次比较时发生的:

ANp≥ANi

此时 p 的初态是由表头 ATO 赋予(ATO→P）的第 1 个元素位置（第 1 个元素是当前表中最小的），自然已就不必和当前表其它元素再作比较了。遍历结束，退出内循环。执行 ai 和 aq。及 ap 勾链操作。于是，特定的操作

ATO→ATi (通过指针 p)

i→ATO (更新表头)

正好使插入元 ai 为新表的第 1 个元素,当前表的第 1 个元素递增为第 2 个元素。

（2）插入元素 ai 的大小介于前驱元 aq 和当前元 ap 之间。此时，提前结束比较遍历，执行勾链操作

i→ATq

p→ATi

使插入元素 ai 及 aq 挂上勾，这种情况的发生是表插入的一般情况。

（3）插入元素 ai 比当前表中全部元素都大。这种情况是在 ai 和当前表中所在兀系逐一比较完时发生的。结束比较遍历的控制条件最末元素的的指针置 0。于是

当 P=0 时退出内循环，插入操作时 ai 作为新表的最后元素，由 P=0，P→ATi，正好把最后元的指针值置为 0，这是勾链操作的另一种特定情况。

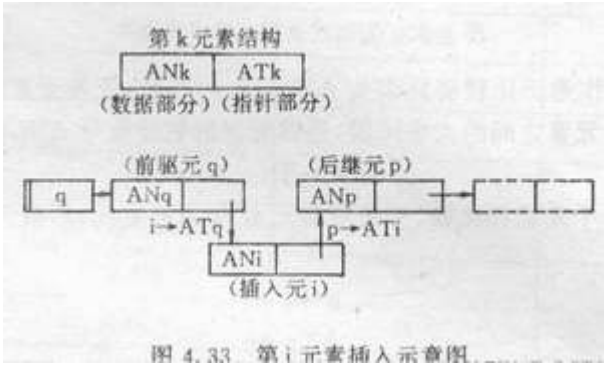


图 4.33 第i元素插入示意图

程序实现的说明：虚设元素 a0 作表头（AN0 为空），可使表插入的三种情况处理具有一致性。当前元素指针 p、前驱元素指针 q、插入元素指针 i 分别取变址器 GR1、GR2、GR3。由于每个元素是由两个单元组成的，因此各元素的序号和它们在内存中的相对位置不是一致的，而使用变址方式访问表元素，必须是内存的相对地址，根据表的存储结构，序号 i 和相对地址的关系是：

<ai>=A+2*i

用元素的序号形成元素的地址的设计方法。使程序结构变成十分紧凑，而且更容易由指针链读出序号而了解序列中各元素的先后次序。

；程序 4．21 勾链式表插入法排序（本程序给定 8 个数据为例）

;A 中表元素为双字长，第一字为数据 ANi，第 M 字为下一元素的位置 ATi，

；A＋0 为空，A＋1 为表头，A＋2 第一个元素…

```
        START
        LEA     GR1,1
        ST      GR1,AT0         ；表头初态指向第一元素
        LEA     GR0,0           ;仅一个元素的指针值应是 0
        SLL     GR1,1
        ST      GR0,AT0,GR1
        LEA     GR3,2           ；自 i=2 开始插 X
LOOP    LD      GR1,AT0         ;表头指向最小的元素位置作为当
                                ；当前元素指针 P 的初值
        LEA     GR2,0           ;前驱元素指针 q 初态为 0
        LEA     GR4,0,GR3       ;保护 i
        SLL     GR3,1           ;双字要求
        SLL     GR1,1
```

；将插入元 ANi 和表元 ANP 及其后继元比较

```
L1      LD      GR0,A,GR1       ;取 ANP
        CPA     GR0,A,GR3       ;ANp：ANi
        JPZ     LINK            ;ANi 比表元小则插入
        LEA     GR2,0,GR1       ;否则，当前元作为前驱,p→q
        LD      GR1,AT0,GR1     ;ATp→p，后继元作为当前元
        SLL     GR1,1
        JNZ     L1              ;后继无指针非零，继续
```

;把新元以勾链式插入在前驱元与后继元之间

```
LINK    ST      GR4,AT0,GR2 ;i→ATq
        SRL     GR1,1
        ST      GR1,AT0,GR3 ;p→ATi
        LEA     GR3,1,GR4   ;i+1→i
        CPA     GR3,CN      ;i<=N?
        JMI     LOOP        ;是,继续插入
        EXIT                ;否则,插入排序完成
CN      DC      9
A       DS      1
AT0     DS      1
        DC      28,0,13,0,25,0,72,0,85,0,39,0,6,0,41,0
        END
```

第 5 章 子程序设计

在上一章我们讨论的是囿于一个程序的自身构造方法。本章所讨论的子程序设计,是指程序之间的关系,或者说一个程序以怎样的构造形式和其他程序相联系,以及由此而引出关于子程序设计的技术和方法。

5·1 子程序基本概念

我们可以从以下三个方面来看，子程序的产生不是偶然的：

1．一个“大程序”由许多部分组成，这各部分的程序段具有相对独立的功能，或者是一个概念清楚的逻辑块。我们可以把这些相对独立的程序段，先用一个功能说明代替它，暂时不去注意它的细节，从宏观上把程序的框架构造出来，使整个程序的逻辑结构清晰。 我们把独立的程序段称为一个模块，各个模块可以一种子程序形式构成，子程序可独立成章。分别编写和调试。这样一个大程序就由

一组独立的、被连接的子程序所组成。主程序实际上只完成一些不断地转移到相应子程序的功能。这种方法大大地简化了整个程序的设计。

2．在一个程序的不同地方，要用到差不多相同结构的程序段，或者在不同的地方，都要执行同一个确定的任务，比如，在几个地方都是求一个数的平方根，差别只是自变量不同，一处需要求$\sqrt{x}$，另一处需要$\sqrt{y}$，等等。通常我们不希望在每个地方都重复地来对这相同任务进行编码。为避免这类重复，可用于程序设计这种代码，它可单独立地放在一个地方，当在程序的某处需要用到它，就转移到该子程序，完成之后，再返回到原来程序调用它的那个地方。

3．任何一种计算机，它的指令系统总是个有限集合。若程序员需要一种指令操作，而这种指令在指令系统中又没有设置，例如，COMET 计算机中没有乘法指令，程序员就可自己设置一条“软”指令 MUL，用 MUL 可以象“硬”指令一样完成乘法的操作。我们用一指令序列，它能完成乘法的运算，并以子程序的形式独立存在。这样，按约定的方式，在程序需要用乘法指令的地方，用一条 CALL MULL 指令，即有等同效果。从指令级的角度看，子程序本身看作是高一级上定义的一条新指令。一旦内存中存储了一个 MUL 乘法子程序，就好比 COMET 计算机扩充了一条乘法指令。

总之，为了节省空间，避免重复编码，简化程序随逻辑结构，采用子程序设计技术是很有必要的。子程序设计是使程序模块化的一种良好的程序设计方法。程序员借助于子程序的设计技术,用“自顶向下”的模块化程序设计方法,开发软件,编写程序。这种程序不仅有良好的程序结构，而且写出的程序易于理解，易于调试，易于维护。

现在我们可以给子程序一个定义，子程序是一组指令，它能完成一定任务，并能在主程序的若干地方调用它；当子程序执行完成后，能自动地返回到主程序调用它的那个地方。

这里所谓的“调用”和“返回”都是程序控制的转移，但不是简单的转移指令。返回一般是指返回到调用它的那条指令的后继指令，而正确的返回，正是在调用它的时候就为返回做好了准备.按照子程序的定义,子程序的设计上满足三个基本条件:

(1)允许控制从一个程序转移到这个程序.

(2)子程序在执行完成后,能自动返回到调用它的那个程序去,并继续运行.

(3)调用程序和被调用程序之间有传递数据的能力.

这就是要讨论的子程序连接问题。字程序的连接,计算机设置有专门的指令来实现。此外，在设计子程序时，要约定信息的传递方式。

子程序在执行过程后，也可以调用其它的子程序，这就是“嵌套”调用。一般我们称调用子程序的程序为主程序，被调用的程序是子程序。因此主程序和子程序也是相对的，主程序和子程序的调用可见图 5．1 的描述。图中主程序 MAIN 调用子程序 A，主程序是调用程序,A 是被调用程序，程序 A 又调用子程序 C。此时 A 是调用程序,C 是被调用程序。相对于 C,A 又可称为主程序。这种情况出现嵌套调用。当然程序 MANI 也可直接调用 C，这时 MAIN 和 C 程序又构成父子关系.

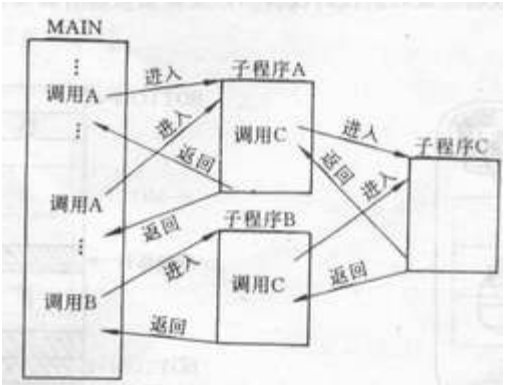


图 5.1 嵌套调用子程序的情形

5.4 调用中的信息传递技术

子程序链接的三个基本条件，以上已论述了两个，即 COMET 计算机设置的转子和返回指令，借助于堆栈机构，实现调用的转移和执行子程序后的返回。余下的一个是主程序与子程序之间的信息传递问题。例如，有一个子程序 SQRT 是计算数的平方根，它要能够适应不同地方来的调用，也要能适应对不同数值的计算。这就必须要求平方根的那个数，由主程序传给 SQRT，并在子程序执行之后，计算的结果应送回调用它的那个程序。传递给子程序的参数，可以是数据的本身，也可以是数据的地址，甚至是数表的地址，假如求出 n 个数的平均值，就可能传递的是这一组数的首址及长度。在论及信息传递时，诸如数据的类型、多少、存储方式各有很大差异，给子程序设计带来一定复杂性。在子程序设计中，参数传递技术是子程序链接的关键问题。因此有必要解决调用程序与被调用程序之间信息传递技术问题，不同情况的传递方法。

本节介绍三种方式传递信息，即约定寄存器，约定调用序列,约定堆栈.

信息的含义可以是控制信息,也可以是数据信息.通常,传递数据信息上区别以下三种形态:数值本身,数值地址,数表的地址.

5．4．1 约定寄存器

约定寄存器方式是调用与被调用的程序之间，传递的参数是放在约定的寄存器中。寄 存器传递信息方式，在设计子程序时被广为使用，因为用寄存器存放信

息，不依赖于具体的内存单元，使子程序具有很强的独立性，尤其是设计外部子程序时，避免了使用外部量及定义全局量的额外开销。同时，递归子程序的设计中，一般都是采用寄存器传递信息的方式。

以下用寄存器方式传递参数，设计求平方根的子程序 SQRT，并以传递数值本身及数值地址两种形态的参数分别得到两种规格的子程序。对不同形态的参数，子程序要区别情况处理，显然主程序要以约定的方式进行调用，达到配合默契。请见以下示例：

主程序给出调用的条件

```
        START
MAIN    LD      GR0,N           ;取 n=>GR0
        CALL    SQRT            ;调用
        ST      GR1,ROOT        ;结果(GR1)存入
        EXIT
N       DC      n               ;自变量 n
ROOT    DS      1               ;放结果
```

子程序 SQRT,(GR0)=n,(GR1)=$\sqrt{n}$

```
SQRT    LEA     GR1,0
        LEA     GR3,1
LOOP    ST      GR3,J
        SUB     GR0,J
        JMI     EXT
        LEA     GR1,1,GR1
        LEA     GR3,2,GR3
        JMP     LOOP
EXT     RET
J       DS      1
        END
```

这里主程序传送给子程序的参数，GR0 中是数据本身。我们也可以用 GR2 传递数据的地址则子程序要稍加改变，如下所示：

```
        START
MAIN    LEA     GR2,N           ;<n>=>GR2
        CALL    SQRT            ;调用
        ST      GR1,ROOT        ;结果(GR1)存入
        EXIT
N       DC      n               ;自变量 n
ROOT    DS      1               ;放结果
```

子程序 SQRT,(GR2)=<n>,(GR1)=$\sqrt{n}$

```
SQUT    LD      GR0,0,GR2       ;取 n=>GR0
```

```
        LEA         GR1,0
        …           (省略部分同上)
EXT     RET
J       DS          1
        END
```

用寄存器传递信息是常见的方式,在变量不多的情况下,约定寄存器是很方便的。以下给出的两个例子都是约定寄存器的方式。

[例 5.1] 正整数乘法子程序。约定被乘数放 GR1，乘数放 GR2，结果放 GR0，且要求（GR0）≤32767。即。

（GR1）*（GR2）=>GR0

解回思想：

①我们还是从低位到高位考察乘数，

当$(GR2)_{15}$= 0 不加被乘数
1 加被乘数

②被乘数（GR）左移 1 位，以便和部分积高位相加，同时乘数（GR2）右移 1 位，转①

③控制乘法运算何时结束，用条件判定控制。因为是正整数，且积不超过 15 位，（GR2）的高位可能为 0，通过右移，只要（GR2）=0。不必再逐位考察下去可以提前结束。以下直接给出［例 5．1］的程序，读者自己画出算法流程图。

：程序 5．1:正整数乖法子程序

```
MUL     START
        LEA     GR0,0       ;积初态为 0
M1      ST      GR1,P       ;保存被乘数
        LEA     GR3,0,GR2   ;保存乘数
        JZE     M4          ;乘数为 0,结束
        AND     GR3,ONE     ;截乘数的末位
        JZE     M3          ;为零,不加被乘数
M2      ADD     GR0,P       ;非零,加被乘数
M3      SRL     GR2,1       ;乘数右移
        SLL     GR1,1       ;被乘数左移
        JMP     M1          ;继续
M4      RET
P       DS      1
ONE     DC      1
        END
```

[例 5.2] 正整数除法子程序。约定被除数放 GR1，除数放 GR2，商放 GR3，余数放 GR1，即：

（GRI）/（GRZ）=>GR3，余数=>GR1

解题思想：

（1）首先判别被除数、除数非负且除数（GR2）≠0，否则除法不进行下去，且以标志-1=>GR3 返回。

（2）因为是正整数除法，故商不超过 15 位。

（3）像通常的算术那样，从高位到低位，用试商的办法，逐次求出商的各位。

（4）操作的步骤是：

①先置部分商 1，当除数小于被除数时，部分商及除数均左移 1 位，再作新的比较以确定某位商。

②作比较时，当发现除数大于被除数时，部分商复位，认定为某位的部分商，同时除数复位，准备做减法。

③被除数减除数，其差作为新的被除数，同时将求出本次的部分商和高位商相拼，作为新高位商。

④重复①②③，直到除数大于被除数且认定的某位部分商已为 0 则除法停止.
读者阅读给定的程序 5．2，并画出其算法流程。

;程序 5．2：正整数除法子程序：

：（GRI）/（GRZ）=>GR3，余数在 GR1 中

```
DIV     START
        ST      GR2,WK1     ;保存除数
        LEA     GR3,-1      ;预置标志-1
        SLL     GR1,0
        JMI     DRET        ;被除数为负
        CPA     GR2,C1
        JMI     DRET        ;除数小于等于零
        LEA     GR3,0
        ST      GR3,SH0     ;商初始化
D10     LD      GR2,WK1
        LEA     GR3,1       ;预置部分商为 1
D20     ST      GR2,WK2
        CPL     GR1,WK2     ;被除数小于除数?
        JMI     D30         ;是,转置商
```

```
        SLL     GR3,1       ;否则,部分商
        SLL     GR2,1       ;及除数左移 1 位
        JMP     D20         ;继续试商
D30     SRL     GR3         ;小于时部分商右移
        JZE     D40         ;部分商为 0 则除法结束
        ADD     GR3,SH0
        ST      GR3,SH0     ;拼成商
        SRL     GR2,1       ;除数右移 1 位
        ST      GR2,WK2
        SUB     GR1,WK2     ;余数(GR1)作为新的被除数
        JMP     D10         ;继续求下个部分商
D40     LD      GR3,SH0     ;商=>GR3
DRET    RET
WK1     DS      1
WK2     DS      1
SH0     DS      1
C1      DC      1
        END
```

5．4．2 约定调用序列

所谓调用序列方式是把要传递的参数和转子指令本身按一定的顺序，形成一个指令调用序列。其中的参数可以跟随在转子指令之后，也可放在转子指令之前。同样，调用序列中的参数可为数值本身，数值地址，数表地址或它们的混合。请见以下示例：

若数据本身跟随转子指令之后，设计一个乘法子程序。主程序的调用方式是

```
MAIN    CALL    MUL
KA      DC      a       ;被乘数及乘数 a,b
KB      DC      b
KC      DS      1       ;存放积
RETP    ……              ;返回点
```

子程序设计中，利用返回迹作为数据的指针将自变量传递给子程序，在子程序求得结果，再传递给主程序.

```
MUL     START
        LD      GR3,0,GR4   ;返回原迹=>GR3
        LD      GR1,0,GR3   ;被乘数=>GR1
        LD      GR2,1,GR3   ;乘数=>GR2
        LEA     GR0,0       ;积初态置 0
M1      ST      GR1,P       ;(GR1)*(GR2)=>GR0
        LEA     GR3,0,GR2
        JZE     M4
```

```
        AND     GR3,ONE
        JZE     M3
M2      ADD     GR0,P
M3      SLL     GR1,1
        SRL     GR2,1
        JMP     M1
M4      POP     GR3             ;积(GR0)送主程序
        ST      GR0,A,GR3
        PUSH    3,GR3
        RET
P       DS      1
ONE     DC      1
```

试问读者，MUL 子程序设计中，为什么不可以直接用

```
   LD   GR1,KA   ;取 a
   LD   GR2,KB   ;取 b
```

将自变量传递给子程序，以及用

```
  ST  GR0,KC       ;把积送入 KC
```

如果调用序列中参数是数据地址，我们可作如下改变:

```
MAIN    CALL   MUL
K       DC     A,B,C     ;被乘数,乘数,积的地址
REPT    ……               ;返回点
```

利用返回迹作为数据地址的指针，实现调用与被调用程序之间的信息传递。读者试给出这
个子程序的连接序列。

5．4．3 约定堆栈

堆栈是共享的缓冲区，调用程序和被调用的程序之间传递信息，也可以藉助于堆栈。堆栈方式一般是在转入子程序之前，由主程序将自变量信息压人堆栈；进入子程序之后，按约定的顺序取出使用，当子程序执行后得到的结果，在执行返回指令 RET 之前压入堆栈，返回主程序后，再由主程序从堆栈中取出结果。这种通过堆栈交换信息称为约定堆栈方式，见如下示例：

```
MAIN   LD       GR2,A
       PUSH     0,GR2     ;被乘数 a 进栈
       LD       GR2,B
       PUSH     0,GR2     ;乘数 b 进栈
       CALL     MUL       ;调用
```

```
K       POP     GR2       ;取积
        ST      GR2,C     ;积存入 C 中
        …
A       DC      a         ;被乘数
B       DC      b         ;乘数
C       DS      1         ;存放积
        …
```

按约定的次序从栈中取出自变量，子程序计算结果再进栈由主程序取回使用

```
MUL   START                    ;先退返回迹=>GR3
      POP         GR3          ;乘数退栈=>GR2
      POP         GR2          ;被乘数退栈=>GR1
      POP         GR1          ;返回迹再进栈
      PUSH        0,GR3
      …
      (GR1)*(GR2)=>GR0
      …
M4    LD          GR3,0,GR4    ;返回迹退出
      ST          GR0,0,GR4    ;积进栈
      PUSH        0,GR3        ;原返回迹放栈顶
      RET
```

子程序连接过程中的栈状态如图 5．9 所示。

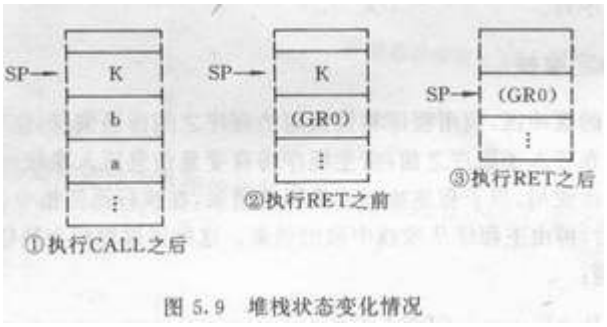


图 5.9 堆栈状态变化情况

约定堆栈传递信息，只要堆栈深度许可的话，一般对传递的信息量可不受限制。读者在应用中应注意的是信息存取的次序不要搞错，既可以用进栈、退栈指令存取信息，也可用堆栈指示器 SP，以变址方式存取栈中的任何一个元素(GP 是寄存器 GR4！)。在进入子程序时，栈顶是返回迹，而不是主程序最后压入栈内的参数，要注意堆栈里内容动态变化，即返回地址和传递信息位置的变化。

主程序和子程序之间传递信息技术主要介绍这三种。根据实际问题的需要，还可以应用其它手段.这三种方式也可以混合使用,不必硬加限制.

5.5 子程序设计应用

本节介绍一些实用子程序的设计例，及模块化程序设计的初步思想.

5．5．1 字符串的十进制数转换为机内数.

[例 5．3] 设计十进制数转换成 M 进制数子程序（10→2）。

①已知十进制数为下列三种形式之一；

a1a2…amB

+a1a2…amB （m≤5，结尾符 B 为非数字字符）

-a1a2…amB

要求数值范围:-32768-+32767

②数字串存储区的首地址放在 GR1 中，转换成二进制数放 GR0 中。

解题思想：通常我们讲的十进制数是指从输入设备上，进入计算机的十进制数字串,每个数字均以字符的 ASC11 码形态存在，必须把它转换成二进制数方可计算。

假定十进制数为

N=a1a2…amB(B 为结束符)

其中 a0 可为“＋”号，“-”号或没有，因此

｜N｜

=a1a2…am

=(…(a1*10+a2)*10+…+a_{m-1})*10+am

只要把数字符 ai(i=1,2,…m)变成二进制值(面值),利用"连乘加"的方法,即可求出｜N｜,遇到任何非数字符即停止乘加计算.其中乘 10,用"移位相加"易得数值的 10 倍值.

在求出｜N｜之后,根据数字中的第一个字符 a0 确定其代数值,当 a0="+",或第一个字符是数字,则 N=｜N｜即得结果,当 a0="-",对｜N｜经过求补操作后,则求出 N 为负数的补码.

当判定 ai 是数字 0 一 9 时，用下式变成二进制值：

'ai'-'0'→ai

其中‘0’是数字 0 的 ASCll 码，其值为 48。

用图 5．10 的流程描述 10→2 的算法。图中用 SIGN 记 N 的正负性,R 存放 mNm，最后的代数值存 GRO。

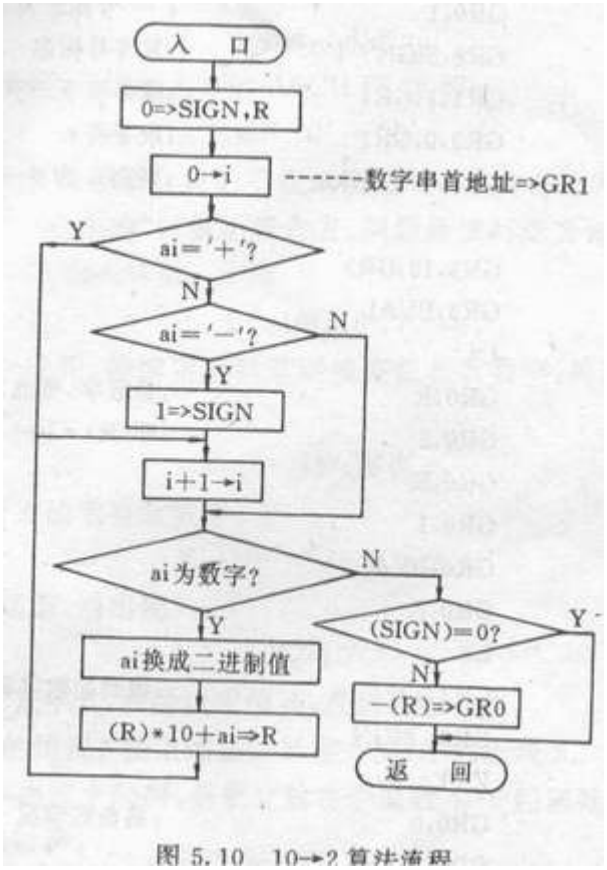


图 5.10 10→2 算法流程

子程序的设计应用约定寄存器方式，（GR1）=数字串首址，假定一个字符数据存放一个单元，则 GR1 传递的是数表地址。程序中应用变址方式，逐一读出各个字符。图中的 i+1→i,即变址器:

(GR1)+1=>GR1

程序的循环控制是以非数字符为终止循环条件。返回时：

（GRO）=被转换的 M 进制数

；程序 5．3：10→2 子程序

；（GRI）=数字串首址，非数字符结束

；（GRO）=二进制数

```
DTB     START
        LEA     GR0,0           ;预置'+'号标志为 0
        ST      GR0,SIGN
        ST      GR0,R           ;结果的初值置 0
        LEA     GR3,43          ;'+'=43
        CPA     GR3,0,GR1       ;第一个字符是'+'号?
        JZE     L1              ;是,转
        LEA     GR3,45          ;'-'=45
        CPA     GR3,0,GR1
        JNZ     L3              第一个字符不是数的符号
        LEA     GR0,1           ;'-'号标志为 1
L1      ST      GR0,SIGN        ;置符号标志
L2      LEA     GR1,1,GR1       ;准备取下一字符
L3      LD      GR3,0,GR1       ;取字符 ai
        LEA     GR3,-58,GR3     ;判别 ai 为 0-9 数字?
        JPZ     L4
        LEA     GR3,10,GR3
        ST      GR3,BVAL
        JMI     L4
        LD      GR0,R           ;是数字,变成二进制值,拼数
        SLA     GR0,2           ;即(R)*10+ai==>R
        ADD     GR0,R
        SLA     GR0,1
        ADD     GR0,BVAL
        ST      GR0,R
        JMP     L2
L4      LEA     GR3,1           ;遇到非数字符则转换完成
        AND     GR3,SIGN
        JNE     EXT
        LEA     GR0,0           ;是负数情况
        SUB     GR0,R
EXT     RET
R       DS      1
SIGN    DS      1
BVAL    DS      1
        END
```

5．5．2 机内 M 进制数转换为十进制数字串

[例 5．4] 设计将二进制数转换成十进制数子程序（2→10）。

①已知二进制数 X 在 GR0 中，存放十进制数字串首地址在 GR2 中。

②要求将 X 转换成如下形式的十进制数字串

dld2…dm 或 -ddld2…dm （m≤5）

即滤去前导零，例如：00409 译成 409 三位数字串。且返回时，数字串长度放 GRO

解题思想：所谓 2→10，是将计算机内的二进制数（它总是某个代数值的补码），转换成十进制数字串，这个数字串是由数的符号与数字字符组成。

对于任何一个二进制数 X(先假定 X≥0)总可以表示成以 10 为基的多项式，即：

$X=d1*10^4+d2*10^3+d3*10^2+d4*10+d5$

显然，用 10^4 去除 X，其商就是 d1，再用 10^3 去除它的余数又可得到 d2，直到上次余封除 10 得到 d4，最后的余数是 d5。这样由多项式系数 di（i=1—5）组成的就是十进制数的字面值：

dld2d3d4d5

将 di 变成数字字符，只要加上 0 的 ASCll 码 48 即：

di+‘0’→’di’

现在讨论如何滤去前导零。我们不能试图用 di=0 舍去的方法，因为有效数字之后 “0”是有意义的，例如 409 中的“0”就不能舍去。问题是如何滤去有效数字前的“0”，因为译出的十进制数最多 5 位，引入常数。

10^4，10^3，10^2，10^1

将 X 与它们逐一比较，能确定何时遇到最高位有效数字，即滤去前导零。例如 X=2345 与 10^4
，因为

$2345<10^4$，滤去 d1

于是我们按以下方法求有效数字：

$X-10^{5-j}$ j=1，2，3，4

j 从 1 开始到 4 结束，当出现

$X\geq10^{5-j}$

时开始遇到有效数字 dj，做除法求值 dj，直到最后一位。

如何处理 X=0 的情况？按上述滤前导零方法，不会出现 $X\geq^{5-j}$,则全部滤去，连数值“0”也看不见了。解决这个问题，是把它放在个位数 d5 一起来处理，不管山是否为 0，均要转换成一个十进制数字。

本例中求 di 的除法用减法代替，即当

①X≥10^{5-j}时，X-10^{5-j}→X

②dj＋1→di 转①

数符的处理：首先判别 X 的正负性，当 X≥0，正号略去，当 X＜0，在转换数字串之前预置一个“-”号，再｜X｜→ X。因此我们只要对非负数 X 给出处理方法即可。图 5．11 给出这个算法的描述。

程序 5．4 中，是重循环结构，其循环体嵌套的是分支结构，用计数控制循环的结束。它的主程序的信息传递是用约定寄存器方式，GR0 是传递数据本身，GR2 是传递数表的地址。进入子程序后，放数字串首址的 GR2 要保护，因为子程序执行过程中修改了 GR2，而在子程序结束时，求数字串的长度时，利用终态与初态之差。

在流程图中 j 的初态为 1，它反映 dj 的顺序及幂常数 SNOj 的值是 SNOj=10^{5-j}。因为数表 SNO 采用 GR1 作指针，按通常习惯是 0=＞GR1 作 SNO1=10^4的变址器值，替代图中的 1→j。

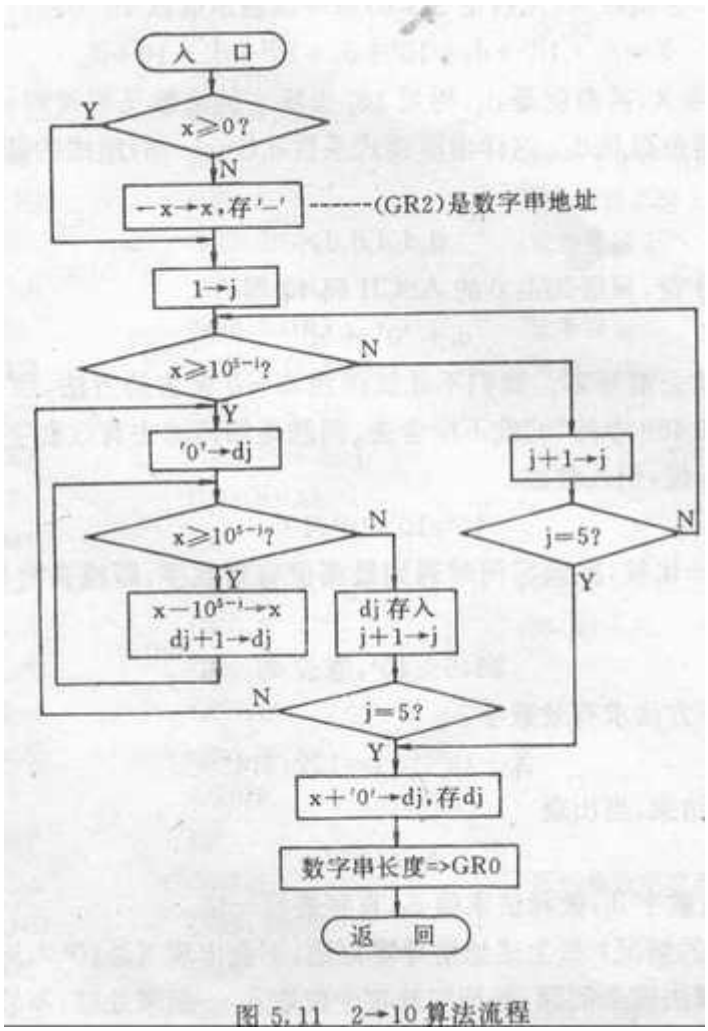


图 5.11 2→10 算法流程

；程序 5．4： 2→10 子程序

；（GR0）=待转换的二进制数 X，（GR2）=存放十进制数字串首址

;返回时(GR0)=数字串长度

```
BTD     START
        PUSH    0,GR2           ;保护首址
        ST      GR0,X
        AND     GR0,X           ;X 是正数
        JPZ     L0              ;是,转
        LEA     GR0,0           ;否则,取绝对值
        SUB     GR0,X
        LEA     GR3,45          ;且送'一'号
        ST      GR3,0,GR2
        LEA     GR2,1,GR2       ;地址递增
L0      LEA     GR1,0           ;1→j(幂常数指针)
L1      CPA     GR0,SN0,GR1     ;X:SN0j
        JPZ     L2              ;大于等于,则准备求 dj
        LEA     GR1,1,GR1       否则,j+1→j,滤去前导零
        LEA     GR3,-4,GR1      ;j=5?
        JNZ     L1              ;不等于,继续
        JMP     L5              ;等于,处理个位数
L2      LEA     GR3,48          ;'0'的 ASCII 码作 dj 的基数
L3      CPA     GR0,SN0,GR1     ;X:SN0j
        JMI     L4              ;<,求下一个数字
        SUB     GR0,SN0,GR1     ;>=,相减,确定 dj
        LEA     GR3,1,GR3
        JMP     L3
L4      ST      GR3,0,GR2       ;存入 dj
        LEA     GR2,1,GR2       ;地址递增
        LEA     GR1,1,GR1       ;j+1→j
        LEA     GR3,-4,GR1      ;j=5?
        JNZ     L2              ;不等于,求下个数字
L5      ADD     GR0,C48         ;处理个位数字
        ST      GR0,0,GR2
        LEA     GR0,1,GR2       ;(GR0)=数字串末址
        SUB     GR0,0,GR4       ;(GR0)=长度(末址-首址)
        POP     GR2             ;恢复数字串首址
        RET
C48     DC      48
```

```
SNO     DC          10000          ;10 的幂常数表
        DC          1000
        DC          100
        DC          10
X       DS          1
        END
```

本例是有符号十进制数的转换，试问读者，若为无符号数的转换（无符号数最大可达 65535），本程序如何修改。

5．5．3 考题选择状况统计

[例 5.5] 设计一个调查上午试题和下午试题的选择状况的子程序。主程序在 GR1 中存放着三个参数表的表地址所在区域的首址，传递给子程序，其格式如下：

```
(GR1)+0   AMTBL 表地址
     +1   PMTBL 表地址
     +2   ANSW 表地址
```

其中：①上午试题的选择状况存放在 AMTBL 为首址的区域内，下午试题的选择状况存放在 PMTBL 为首址的区域内。每个考生的状况占 2 个字。不过，同一考生不一定都参加了上午考试和下午考试，也就是说，同一考生不一定在上午试题选择状况表和下午题选择 状况表中，都存在该考生的选择状况数据。

②AMTBL 表及 PMTBL 表的长度至少为 3 个字，表尾单元中存放-1。每一考生选择状况数报是:第 1 个字中存放考生的报名号,第 2 个字的第 0 位至 15 位表示了上中试题 1—16 题的选择状况(相应位为 1 时，表示选择了该题；为 0 时，表示未选择该题).选择状况表按报名号顺序排列。上下午考生选择状况的数据格式相同。

③ANSW 表是考题选择的人数统计表。ANSW 表由连续的 16 个单元组成，依次对应 1—16 个考题。

编制子程序要求：按 AMTBL 表及 PMTBL 表给定的信息，统计出同时选作了上午试题和下午试题的各题人数，分别送入 ANSW 表中的相应单元。

解题思想：本题的说明较长，但不难理解，首先是耐心审题意，就可大致确定算法。现 对图 5.12 给出的程序流程说明如下：

（1）为了便于理解算法，图 5．12 将主程序传递给子程序的信息，引用原来主程序中定义的名字。

（2）每个考生在 AMTBL 表及 PMTBL 表中，占有的信息是如下两个字：

因为一个考生不一定同时都参加了上午考试及下午的考试，因此在两张表中，找出有相同报名号的,才是同一考生.

考生报名号
选题状况

（3）虽为同一考生，但他的上午选题状况和下午选题状况是不尽相同的。例如，某考生对上、下午 16 题选择状况如下两个字表示：

上午

0	1	2	13	14	15
1	1	1…1		0	0

下午

0	1	1…1	1	0

可见上午考题，除 15、16 两题未选外，其余 14 个题均选做了，下午是第 1 题和第 16 题未选。那末，该考生上午和下午同时选择的是第 2 至第 14 题，用以下操作即可得该考生的选择结果：

上午∧下午=>GRO

（4）得到一个考生的选择结果之后，可进行数据统计。只要当 GRO 中有 1 的位则 ANSW 表中对应单元内容加 1，否则不加，本题用另一个子程序 INCR 实现。用 GR3 放统计表首址 ANSW，GRO 放考生的选择结果，传递给子程序 INCR。

（5）本程序是条件控制的循环结构。因为两个表的结尾是-1 标志，任何一个表中扫描遇到一 1，整个循环都环应结束。

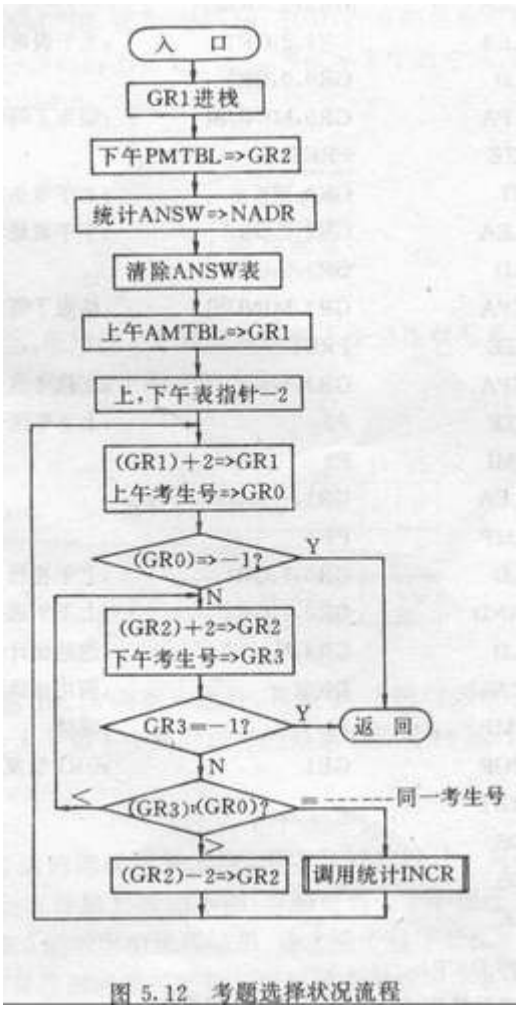


图 5.12 考题选择状况流程

;程序 5.5: 上下午选题统计

```
PTNCHK   START
         PUSH    0,GR1          ;GR1 进栈
         LD      GR2,1,GR1      ;下午状况表首址
                                PMTBL=>GR2
         LD      GR3,2,GR1      ;选题统计表首址
                                ANSW==>GR3
         ST      GR3,NADR       ;送 NADR 暂存
         LEA     GR0,0          ;清除选题统计表 ANSW
         LEA     GR3,16,GR3
CLER     LEA     GR3,-1,GR3
         ST      GR0,0,GR3
         CPL     GR3,NADR
         JNZ     CLER
         LD      GR1,0,GR1      ;上午状况表首址
                                AMTBL=>GR1
         LEA     GR1,-2,GR1
```

```
        LEA     GR2,-2,GR2
P1      LEA     GR1,2,GR1       ;上午地址表增 2
        LD      GR0,0,GR1
        CPA     GR0,MINUSI      ;结束了吗?
        JZE     PRET
        ST      GR0,WK          ;上午考生号
P2      LEA     GR2,2,GR2       ;下午地址表增 2
        LD      GR3,0,GR2
        CPA     GR3,MINUSI      ;结束了吗?
        JZE     PRET
        CPA     GR3,WK          ;比较考生号
        JZE     P3              ;上下午同一考生,则统计
        JMI     P2
        LEA     GR2,-2,GR2
        JMP     P1
P3      LD      GR0,1,GR1       ;上午答题
        AND     GR0,1,GR2       ;上下午选择结果送 GR0
        LD      GR3,NADR        ;选题统计表地址>GR3
        CALL    INCR            ;调用选题统计子程序
        JMP     P1              ;继续
PRET    POP     GR1             ;GR1 恢复
        RET
NADR    DS      1
WK      DS      1
MINUSI  DC      -1
;选题统计子程序 INCR
;(GR0)=考生选题结果,(GR3)=选题统计表首址
INCR    PUSH    0,GR1
SHF     ST      GR0,WK
        SLL     GR0,0
        JZE     GRET
        LEA     GR1,1
        AND     GR1,WK
        ADD     GR1,15,GR3
        LEA     GR3,-1,GR3
        SRL     GR0,1
        JMP     SHF
SRET    POP     GR1
        RET
        END
```

第 6 章 试题精选

本章所选的试题，绝大部分是日本近年来软件水平考试 CASL 汇编语言的试题。有一部分是日本或国内曾以 CAP 汇编语言写的试题，经过改编后列出的。

如果仅仅是听听、看看本书是很不够的，只有动手实践才能学到方法和技巧。基于这个原因，选在这章的试题没有直接给出答案。

6.1 控制两数操作的程序

本程序是一个对两个正整数进行指定操作的子程序。主程序把要进行操作的两数存放在 GR1 和 GR2 中，并在 GR3 中给出所要进行操作的编码（见下表）然后转入子程序，子程序在 GR0 中给出操作结果，再返回主程序，且要求 GRI—GR3 内容不被破坏.

(GR3)	对(GR1),(GR2)的操作
0	(GR1)+(GR2)
1	(GR1)-(GR2)
2	(GR2)-(GR1)
3	MAX((GR1),(GR2))
4	MIN((GR1),(GR2))

[问题]在程序的(1)-(7)七行中各填入一条正确的指令，以完成此程序。除非必要，标号栏不必填写。

[程序]

```
SUBRTN  START
                (1)
        ST      GR2,WORK
        LEA     GR0,0,GR1
                (2)
        JMP     0,GR3
LB0     ADD     GR0.WORK
        JMP     LB6
LB1     SUB     GR0,WORK
                (3)
LB2     LD      GR0.WORK
                (4)
LB3     SUB     GR0,WORK
        JMP     LB6
```

```
        SUB     GR0,WORK
        LD      GR0,WORK
                (5)
        JMP     LB5
LB4     SUB     GR0,WORK
        LD      GR0,WORK
                (6)
LB5     LEA     GR0,0,GR1
                (7)
        RET
LTBL    DC      LB0
        DC      LB1
        DC      LB2
        DC      LB3
        DC      LB4
WORK    DS      1
        END
```

6.12 数字串前导零的设置

[程序说明]

本程序是将输入的 1—15 位数存于由 OUTBUF 开始的 15 个字的右面(见下例示)

（1）若输入的数据超过 15 位，则要求重新输入。

（2）若输入的数据不足 15 位测在输出时插入前导 0。

[例]在输入是 9 位的情况：

输入数据（INBUF）：

```
1 2 3 4 5 6 7 8
9
```

输出数据（OUTBUF）：

```
0 0 0 0 0 0 1 2 3 4 5 6
7 8 9
```

插入前导 0

（3）约定数字以外的字符不得输入，并以回车符终止输入。

[问题]为完成本程序在（1）一（4）行中填上正确的指令。

[程序]

```
PROG2   START     BEGIN
BEGIN   IN        INBUF,LENG      ;数据输入
        LEA       GR0,0
        CPA       GR0,LENG
        JZE       LAST
        LEA       GR0,15
        CPA       GR0,LENG        ;个数≤15?
                  (1)             ;是的
        OUT       ERMSG,C9        ;不是
        JMP       BEGIN
GOOD    LD        GR1,LENG
                  (2)             ;输入数据的指针
        LEA       GR2,14          ;输出数据的指针
LOOP1   CPA       GR1,BZERO       ;输入处理完了?
        JMI       SKIP1
        LD        GR3,INBUF,GR1
                  (3)
        LEA       GR1,-1,GR1
        LEA       GR2,-1,GR2
        JMP       LOOP1
SKIP1   LD        GR3,CZERO
LOOP2   CPA       GR2,BZERO
        JMI       SKIP2
        ST        GR3,OUTBUF,GR2
                  (4)
        JMP       LOOP2
SKIP2   OUT       OUTBUF,C15
        JMP       BEGIN
LAST    EXIT
LENG    DS        1
INBUF   DS        80
C15     DC        15
OUTBUF  DS        15
C9      DC        9
ERMSG   DC        'ERMESSAGE'
CAERO   DC        '0'
BZERO   DC        0
        END
```

试题讲解

1. 软件基础知识试题精解

高级程序员级软件基础知识试题覆盖面宽，有一定的理论深度，需要考生全面系统地掌握大
纲所规定的内容。数据结构、操作系统、程序语言、软件工程是考核的重点。二叉树、图、
存储管理、编译原理、软件设计方法等知识点考查的次数比较多，应当是重点中的重点。

1990～2000 年软件基础知识试题，按所涉及的知识点统计如下：数据结构基础知识 14 道题，
程序语言基础知识 13 道题，操作系统基础知识 17 道题，软件工程 21 道题，数据库基础知 13
道题，多媒体基础知识 5 道题。

2.1 数据结构基础

2.1.1 主要知识点

掌握线情表、多维数组、阵列、栈、树、二叉树，图的定义、存储和操作以及常见的排序和查找算法。重点是二叉树和图以及与其相关的算法。

2.1.1.1 数据结构概述

数据结构是指数据对象及其相互关系和构造方法，一个数据结构 B 形式上可以用一个二元组表示为 B=(A，R)。其中，A 是数据结构中的数据(称为结点)的非空有限集合，R 是定义在 A 上的关系的非空有限集合。数据结构按逻辑关系的不同分为线性结构和非线性结构两大为，共中非线性结构又可分为树形结构和图结构，树形结构又可分为树结构和二叉树结构。

2.1.2 线性表

数据结构中，线性结构习惯称为线性表。线性表是最简单也是最常用的一咱数据结构，它是由相同类型的结点组成的有限序列。一个由 n 个结点 e_0，e_1，…e_{n-1} 组成的线性表记为(e_0，e_1，…e_{n-1})。线性表的结点个数称为线性表的长度，长度为 0 的线性表称为空的线性表，简称空表。对于非空线性表，e_0是线性表的第一个结点，e_{n-1}是线性表的最后一个结点。线性表的结点构成一个序列，对序列中两个相邻结点 e_{i-1}和 e_i，称前者是后者的前驱结点，后者是前者的后继结点。线性表最重要的性质是线性表中结点的相对位置是确定的。

线性表的结点也称为表元，或称记录，要求线性表的结点是同一类型的任何数据。线性表的结点可由若干个成分组成，其中能唯一标识表元的成分称为了，简称键。线性表包含的表元个数可以动态增加或减少，可以在任何位置插入或删除表元。

线性表常用的运算可分成 4 类：查找运算、插入运算、删除运算和其他运算。每类又包括若干种运算。如查找运算就有两种，一种是查找线性表中某个结点的值，另一种是在线性表中查找具有给定键值的记录。

有多种存储方式能将线性表存储在计算机内，其中最常用的是顺序存储和链接存储。

(1)线性表的顺序存储

线性表的顺序存储是最简单的存储方式。程序通常使用一个足够在的数组，从数组的第一个元素开始，将线性表的结点依次存储在数组中。用数组元素的顺序存储来体现线性表中结点的先后次序关系。其最大优点是能直接访问线性表中的任意一个结点。

(2)线性表的链接存储

用单链表存储线性表是线性表的链接方存储方式。从链表的第一个表元开始，将线性表的结点依次存储在链表的各表元中。链表的每个表元除了要存储线性表结点的信息外，还要有一个成分用来存储其后继结点的指针。

2.1.1.3 栈和队列

栈是只允许在同一端进行插入和删除运算的线性表。允许插入和删除的那一端称为栈顶，另 一端为栈底。若有栈 $S=(s_0, s1, \cdots, s_{n-1})$，则 s0 为栈底结点，$s_{n-1}$ 为栈顶结点。习惯称插入栈的结点为进栈，删除栈的结点为出栈。因为最后进栈的结点必定最先出栈，所以栈具有后进先出的的特性。可以用顺序存储线性表来表示栈，栈也可以用链表实现，用链表实现的栈称为链接栈。

队列是只允许在一端进入插入，另一端进行删除运算的线性表。允许删除运算的一端称为队首，允许插入运算的端为队尾。习惯称插入队列结点为进队，删除队列结点为出队。若有队列 $Q=(q0, q1, \cdots, q_{n-1})$ q0 为队首结点，q_{n-1} 为队尾结点。因最先进入队列的结点将最先出 队，所以队列具有先进先出的特性。可以用顺序存储线性表来表示队列，也可以用链表实现
，用链表实现的栈称为链接队列。

2.1.1.4 数组和字符串

数组是最常用的数据结构之一，一般用来顺序存储线性表。数组由固定个数的元素组成，全部元素的类型相同，元素依次顺序存储。每个元素对应一个下标，数组元素按数组名和元素的下标引用，数组元素的下标个数称为数组的维数。在 C

语言中，约定数组的第 1 个元素的下
标为 0，其余依次类推，由 n 个元素组成的数组，其最后一个元素的下标为 n-1。数组元素可
以是任何类型的，当元素本身又是数组时，就构成多维数组。

多维数组是一维数组的推广，多维数组中最常用的是二维数组。多维数组的所有元素并未排
在一个线性序列里，要顺序存储多维数组就需要按一定次序把所有的数组元素排在一个线性
序列里，常用的排列次序有行优先顺序和列优先顺序两种。对于多维数组，C 语言按行优先
顺序存放。

字符串是非数值处理应用中重要的处理对象。字符串是由某字符集上的字符所组成的任何有
限字符序列。不包含任何字符的字符串称为空字符串。字符串所包含的有效字符数称为字符
串长度。一个字符串中任意连续的子序列称为该字符串的子串。

2.1.1.5 树

树和二叉树是非线性数据结构，用它们能很好地描述有分支和层次特性的数据集合。习惯称
树和二叉树数据结构为树形结构。在许多算法中，都经常使用树形结构描述问题的求解过程
或表示求解的对策等。由于二叉树有确定的分支个数，又可以为空，有良好的递归性质，与
一般树相比较，特别适宜于程序设计，因此也常将树转换成二叉树进行处理。

(1)树的基本概念

树是一种多分支多层次数据结构，由一组结点组成。由于它呈现出与自然界的树类似的结构
形式，所以称它为树。树是由一个或多个结点组成的有限集 T，它满足以下两个条件：

①有一个特定的结点，称为根结点；

②其余的结点分成(m＞m=0)个互不相交的有限集 T0，T1，…，Tm-1。其中每个集合又都是一
棵树，称 T0，T1，…，Tm-1 为根结点的子树。

树的定义是递归的，即一棵树由子树构成，子树又由更小的子树构成。一棵树至少有一个结

点。一个结点的子树数目，称为结点的度。树中各结点的度的最大值称为树的度。

(2)树的存储结构

树是非线性的结构，不能简单地用结点的线性表来表示。树有多种实用的存储结构，最常用
是标准存储结构和带逆存储结构。在树的标准存储结构中，树中的结点内容可分成两部分
：结点的数据和指向子结点的指针数组。当程序需从结点返回到其父结点时，需要在树的结
点中存有其父结点的位置信息，这种存储形式就是带逆存储结构。

(3)树的遍历

在应用树结构时，常要求按某种次序获得树中全部结点的信息，这可通过树的遍历操作来实
现。常用的树的遍存方法有：

·树的前序遍历。首先访问根结点，然后从左到右按前序遍历根结点的各棵子树。

·树的后序遍历。首先从左到右按后序遍历根结点的各棵子树，然后访问根结点。

·树的层次遍历。首先访问处于 0 层上的根结点，然后从左到有依次访问处于 1 层、2 层上的
结点等，即自上而下从左到右逐层访问树各层上的结点。

·访问树中的所有叶子结点。

2.1.1.6 二叉树

(1)二叉树的基本概念

与一般的树结构相比较，二叉树在结构上更规范、更有确定性，二叉树的每个结点有两棵子
二叉树，分别简称为左子树和右子树。因二叉树可以为空，所以二叉树中的结点可能没有子
结点。二叉树的定义为：二叉树是一个有限的结点集合，该集合或者为空，或者由一个根结
点及其两棵互不相交的、左右二叉子树所组成。

二叉树与树不同，首先二叉树可以为空，空的二叉树没有结点。另外，在二叉树中，结点的

子树是有序的，分左、右两棵子二叉树。一般情况下，二叉树常采用类似树的标准存储形式
来存储。

(2)二叉树的遍历

树的所有遍历方法都适用于二叉树，常用的二叉树遍历方法有 3 种：

• 前序遍历。访问根结点→按前序遍历根结点的左子树→按前序遍历根结点的右子树

• 中序遍历。按中序遍历根结点的左子树→访问根结点→按中序遍历根结点的右子树

• 后序遍历。按后序遍历根结点的左子树→按后序遍历根结点的右子树→访问根结点

注意以上 3 种遍历方法都是递定义的。

2.1.1.7 二叉查找树

查找树便于链式存储，还能实现快速查找。作为一种特殊的二叉树，它或者为空，或者满足
下列条件：

• 若该树根结点的左子树非空，其左子树所有结点的键值都小于该树根结点的键值。

• 若该树根结点的右子树非空，其左子树所有结点的键值都大于该树根结点的键值。

• 该树的根结点的左子树和右子树均为查找树。

根据以上定义，如果进行中序遍历，即可得到一个从小到大的结点序列。

2.1.1.8 图

(1)图的基本概念

图是比较树形结构更复杂的一种数据结构。在线性结构中，除首结点没有前驱结点，末结点
没有后继结点之外，一个结点只有一个前驱结点和一个后继结点。在树形结构中，除根结点
没有前驱结点外，一个结点虽只有一个前驱结点，但可以有多个后继结点。但在图结构中，
一个结点的前驱结点和后继结点的个数是任意。

在图中，数据结构中的结点被称为顶点，结点之间的关系被称为边。一个图 G 由非空有限的
顶点集合 V 和有限的边的集合 E 组成，记为 G=(V，E)。

图分有向图和无向图。图中代表边的结点的偶对如果是有序的，则称此图为有向图。在有向
图中用(V1，V2)表示一条有向边，V1 称为边的起点，V2 称为边的终点。(V1，V2)和(V2，V1)
代表不同的边。图中代表一条边的结点的偶对如果是无序的，则称此图为无向图。在无向图
中，(V1，V2)和(V2，V1)这两个偶对表示同一条边。

如果限定任何一条边或弧的两个顶点都不相同，一个有 n 个顶点的无向图至多有 n(n-1)/2 条
边，这样的无向图称为无向完全图。一个有向图至多有 n(n-1)条弧，这样的有向图称为有向
完全图。如果给图的每条边赋予一个实数作为边的权，则该图称为带权图，或称为网。

(2)图的存储结构

最常用的图的存储结构是邻接矩阵和邻接表。

图的邻接矩阵是反映顶点间邻接关系的矩阵，设 G(V，E)是具有 n(n≥1)个顶点的图，G 的邻
接矩阵 M 是一个 n 行 n 列的矩阵，若(i，j)或(i，j) E，则 M［i］［j］=1；否则 M［i］［j］
=0。由邻接矩阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵不定的对称。
图也可用邻接表来存储，为图的每个顶点建立一个链表。且第 i 个链表中的结点表示与顶点 i
相关联的一条边，或由顶点 i 出发的一条弧。有 n 个顶点的图，需要这 n 个链表的头指针按顺
序线性表方方式存储。在无向图的邻接表中，对应某结点的链表的结点个数就是该顶点的度
。

(3)图的遍历

图的遍历是指从图中的某个顶点出发，沿着图中的边或弧访问图中的每个顶点，并且每个顶
点只被访问一次。图的遍历通常采用深度优先搜索或广度优先搜索方法。

图的深度优先搜索类似于树的前序遍历。从图中某个结点出发，访问此结点，然后依次从此

结点未被访问的邻接点出发进行深度优先搜索，直至图中所有该结点有路径相通的结点均被 访问到。若此时图中尚有结点未被访问，则另选图中一个未被访问的结点作起始点，重复上
述过程，直至图中所有结点都被访问到为止。

广度优先搜索类似于树的层次遍历。从某个结点出发，访问此结点，然后依次访问与此结点
邻接的、未被访问过的结点，然后再分别从这些为出发进行广度优先周游，直至图中所有被
访问过的结点的相邻结点都被访问到。若此时图中有结点尚未被访问，则另选图中一个未曾
访问过的结点作为起点，重复上述过程，直至图中所有结点都被访问到为止。

(4)最小代价生成树

设 G=(V,E)是一个连通的无向图，若 G1 是包含 G 中所有顶点的一个无回路的连通子，则称 G1 为 G 的一棵生成树。含有 n 个顶点的连通图的生成树有 n 个顶点和 n-1 条边。对一个带权的图(网)
，在一棵生成树中，各条边的权值之和称为这棵生成树的代价。一个图可以有许多不同的生成树，其中代价最小的生成树称为最小代价生成树。

(5)求最短路径

求最短路径就是从图中某个顶点到其他顶点的最短路径。基本思想是把图中所有结点分成两
组，第一组包括已确定最短路径的结点，第二组包括尚未确定最短径的结点，按最短路径长
度递增的顺序逐个把第二组的结点加到第一组中去，直至从某结点出发可以到达的所有结点
都包括到第一组中。在这个过程中，总保护从该结点到第一组各结点的最短路径长度都不大
于从该结点到第二组的任何结点的最短路径长度。

另外，拓扑排序和计算关键路径都是有向图的重要运算。

2.1.1.9 排序与查找

对于有 n 个结点的线性表(e_0，e_1，…e_{n-1})，按照结点中某些数据项的值递增或递减的次
序，重新排列线性表结点的过程，称为排序。排序时参照的数据项称为排序码，通常选择结
点的键值作为排序码。在排序过程中，线性表的全部结点都在内存、并在内存中调整它们在
线性表中的存储顺序，称为内排序。在排序过程中，线性表只有部分结点被调入内存、并借

助内存调整结点在外存中的存放顺序的排序方法称为外排序。常用的排序方法有：选择排序
、直接插入排序、冒泡排序、希尔排序、堆排序、快速排序、合并排序和外排序，其中外排
序是对大文件的排序。

查找就是在按某种数据结构存储的数据集全中，找出满足指定条件的结点的过程。按查找的
条件分类，有按结点的关键码查找、按关键码以外的其他数据项查找和按关键码以外的其他
数据项的组合查找等。按查找数据在内存还是在外存分为内存查找和外存查找。按查找的目
的分类，如果查找只是为了确定指定条件的结点存在与否，称为静态查找。如果查找是为确
定结点的插入位置或为了删除找到的结点，称为动态查找。

散列表又称杂凑表，是一种非常实用的查找技术。由于查找码与结点在数据结构中的位置之
间不存在确定的关系，查找只能通过查找码与结点的关键码的反复比较来实现。对于通过比
较来缩小查找范围来说，唯一能提高查找效率的办法是通过一次比较能大幅减小查找范围。
最理想的情况是直接利用查找码的信息，一次或几次存取便能存取所查结点。为此必须在结
点的存储位置和它的关键码间建立一个确定的关系，从而让查找码直接利用这个关系确定结
点的位置。

2.1.2 试题解析

数据结构是每年必考的知识点，与初级程序员级考试和程序员级考试相比，高级程序员级考
试涉及的内容具有一定的广度和深度。从历年试题统计(见表 2-1)来看，考查的内容主要是
树、二叉树、图和排序算法，特别是排序算法的平均比较比较次数，考查的次数相当多，复
习是应作为重点，并应熟练掌握相关的算法。

表 2-1 历年高级程序员级数据结构基础试题统计

试题	考查知识点
1990 年试题 4	查找算法的平均查找长度（平均比较次数）
1990 年试题 6	正则二叉树
1991 年试题 3	树和二叉树
1992 年试题 6	常用排序算法
1993 年试题 6	有向图及其遍历
1994 年试题 15	常用排序算法的平均比较次数
1995 年试题 3	树和二叉树
1996 年试题 1	二叉树及其遍历
1996 年试题 14	图和树的判读
1997 年试题 4	线性表、二叉树、树、图的结构及其遍历方法
1998 年试题 3	常用排序算法
1999 年试题 1	常用排序算法
1999 年试题 2	图的概念及图的存储结构
2000 年试题 1	树和二叉树

试题 1(2000 年试题 1)

从供选择的答案中，选出应填入下面叙述中 {}内的最确切的解答，把相应编号写在答卷的对应栏内。

二叉树的前序、中序和后序遍历法最适合采用{ A }来实现。

查找树中，由根结点到所有其他结点的路径长度的总和称为{ B }，而使上述路径长度总和达到最小的树称为{ C }，它一定是{ D }。

在关于树的几个叙述中，只有{ E }是正确的。

供选择的答案

A： ①递归程序 ②迭代程序 ③队列操作 ④栈操作

B： ①路径和 ②内部路径长度 ③总深度 ④深度和

C： ①B-树 ②B+-树 ③丰满树 ④穿线树

D： ①B-树 ②平衡树 ③非平衡树 ④穿线树

E：①用指针方式存储有 n 个结点的二叉树，至少要有 n+1 个指针

②m 阶 B-树中，每个非叶子结点的后件个数≥ [m/2]

③m 阶 B-树中，具有 k 个后件的结点，必含有 k-1 个键值

④平衡树一定是丰满树

【解析】

本题综合考查树形数据结构知识。部分内容与 1991 年试题 3 相似。

由于二叉树的前序、中序和后序遍历方法都是递归定义，所以最适合采用递归程序来实现。

查找树中，由根结点到所有其他结点的路径长度总和称为内部路径长度。具有最小内部路径
长度的树是丰满树，对丰满树的查找树进行插入或者删除操作后，会产生一棵非丰满树。平
衡二叉树是指其上任一结点的左右子树的高度(或者结点个数)保持一定比例的树，即平衡树
上任一结点的左、右子树仍然保持平衡。平衡树的查找效率和丰满树相近，但是在插入或者
删除结点时，更能动态地平衡树的特点。由丰满树同平衡树定义可知，丰满树一定是平稳树
，但是平衡树不一定是丰满树。

m 阶 B-树是一种平衡的 m 叉树，具有如下的性质：

(1)每个结点的后件个数小于等于 m；

(2)除了根结点的各结点之外，每个结点的后件个数大于等于(m/2)；

(3)具有 k 个后件的非叶结点含有 k-1 个键值；

(4)所有叶结点在同一层上，而且不附有信息。

【答案】：A：① B：② C：③ D：② E：③

试题 2(1999 年试题 1)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

结定结点的关键字序列(F，B，J，G，E，A，I，D，C，H)。对它按字母的字典顺序进行排列
，采用不同方法，其最终结果相同，但中间结果是不同的。

Shell 排序的第一趟扫描(步长为 5)

结果是结果应为{ A }。

冒泡排序(大数下沉)的第一趟起泡的效果是{ B }。

快速排序的第一趟结果是{ C }。

二路归并排序的第一趟结局是{ D }。

若以层次序列来建立对应的完全二叉树后，采用筛选法建堆，其第一趟建的堆是{ E }。

供选择的答案

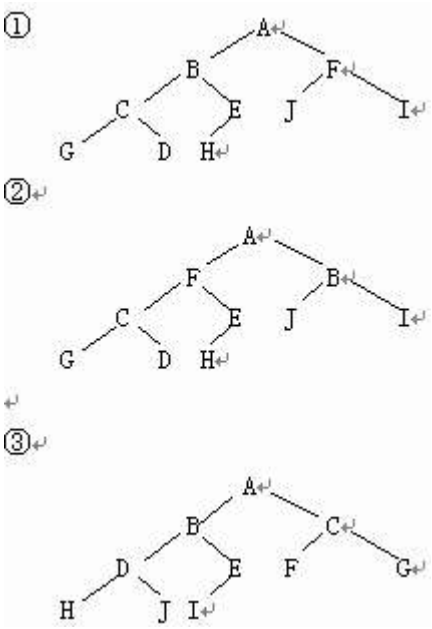
A：①(B，F，G，J，A，D，I，E，H，C)
②(B，F，G，J，A，E，D，I，C，H)
③(A，B，D，C，E，F，I，J，G，H)
④(C，B，D，A，E，F，I，G，J，H)

B：①(A，B，D，C，F，E，I，J，H，G)
②(A，B，D，C，E，F，I，H，G，J)
③(B，F，G，E，A，I，D，C，H，J)
④(B，F，G，J，A，E，D，I，C，H)

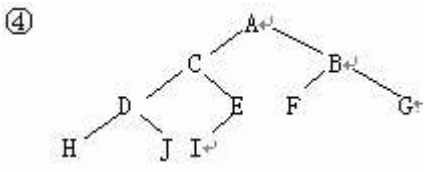
C：①(C，B，D，A，F，E，I，J，G，H)
②(C，B，D，A，E，F，I，G，J，H)
③(B，A，D，E，F，G，I，J，H，C)
④(B，C，D，A，E，F，I，J，G，H)

D：①(B，F，G，J，A，E，D，I，C，H)
②(B，A，D，E，F，G，I，J，H，C)
③(A，B，D，C，E，F，I，J，G，H)
④(A，B，D，C，F，E，J，I，H，G)

E：



④



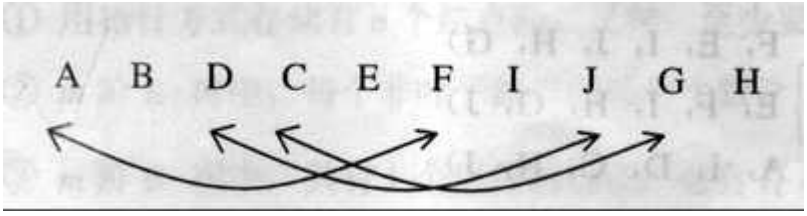
【解析】

解答本题的关键是要掌握几种主要的排序算法。排序是数据处理中经常使用的一种重要运算
。插入排序、选择排序、交换排序、基数排序和归并排序是几种常用的排序方法，每种方法
中还包括更具体的方法。

插入排序的基本思想是：每一步将一个待排序的记录按其关键码值的大小插入到前面已排序
的文件中的适当位置上，直到全部插完为止。Shell 排序是插入排序的一种，按增量将待排
序记录分组，先取增量 i＜n，把全部记录分成 i 个组，所有距离为 i 的倍数的记录放在一组中
，各组内采用插入顺序。然后取 j＜i 作为增量，重复上述分组和排序工作，直至增量为 1，
即所有记录放在一个组中排序为止。题中步长 i=5，先将文件分为 5 组，如下所示：



然后在组内用插入法排序，进行第 1 趟扫描，结果如下，只有用箭头连接的组排序位置发生
变化。



交换排序的基本思想是两两比较待排序记录的关键码值，并交换不满足顺序要求的那些偶对
，直到全部满足为止。冒泡排序和快速排序又是交换排序的两种具体方法。冒泡排序将待排
序的记录两两比较，若为逆序则交换。在升序排序过程中，关键码大的记录下沉，

关键码小
的记录上浮，故称冒泡。将序列按照此方法不到尾处理一遍后，当前处理范围内最大的记录
被交换到最后，如此往复，最后得到一个有序的记录序列。问题 B 比较简单，顺次将关键字
序列两两比较，第 1 趟处理结果显然选③。

快速排序是对冒泡排序的一种改进，方法是：在待排序序列中确定一个记录，以它为基准，
用交换的方法将所有的记录分成两部分，即关键码值比它小的一部分和比它大的一部分。再
分别对两个部分实施上述过程，一直重复到排序完成。问题 C 解答比较简单。取第一个记录 F
为基准，将比 F 大的移到后面，比 F 小的移到前面。一般采用从两端往中间加入的移动方式，

第 1 趟排序的过程：
F，B，J，G，E，A，I，D，C，H
C，B，J，G，E，A，I，D，F，H
C，B，F，G，E，A，I，D，J，H
C，B，D，G，E，A，I，F，J，H
C，B，D，F，E，A，I，G，J，H
C，B，D，A，E，F，I，G，J，H--第一趟结束

归并排序要求待排序列已经部分排序，形成了若干子序列。归并排序就是将已排序的子序列
进行合并，得到完全排序的序列。合并时比较各子序列的第 1 个记录，便可找出第 2 个记录。
如此继续，一遍扫描即可得出结果。具体的方式分为二路归并和多路归并。采用二路归并对
有 n 个记录的序列进行排序时，应先将序列分成 n/2 个子序列，使每个子序列包含 2 个(或 1 个)
记录，在此子序列内进行排序。然后再将记录序列分成 n/4 个子序列(即从前往后两两合并子
序列)，再在子序列内进行排序。重复上述过程，直到只有一个子序列，则排序完成。理解
了这个过程，则问题 D 的解答就会非常简单。先将待排序列(F，B，J，G，E，A，I，D，C，H
)两两合并，分成 5 个子序列，在各子序列内排序，第一趟结局显然为(B，F，G，J，A，E，D
，I，C，H)。

选择排序的基本思想是每次从待排序的记录中选择出码值最小(或最大)的记录，顺序放在已
排序的记录序列的最后，直到全部排完为止。堆排序是选择排序的一种，采用堆

排序时，首
先要按层次序列来建立对应的完全二叉树，然后按筛选法建立立堆。在建堆的过程中，要记
住堆的特征，作为堆的二叉树具有这样的特征：树根的关键码值小于树叶的关键码值；堆的
调整过程是从最下层开始的，先将二叉树的树根与左子树进行比较，如果树叶小于树根，则
交换；然后比较树根与右子，若树叶小于树根，则交换。同时，在交换的过程中如果影响到
下层的堆，需要同时调整下层的堆。题中 E 问题的解答用示意图说明。在这里，初始建成的
完全二叉树如图 2-1(a)所示。在调整完最下层的子树后，整个子树的结构如图 2-1(b)所示。
调整完第 2 层后，整个二叉树的结构如图 2-1(c)所示。最后一次调整的结果就是题中问题 E 的
答案②。

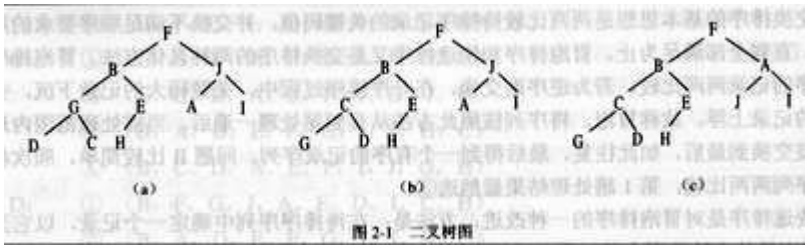


图 2-1 二叉树图

【答案】：A：③ B：③ C：② D：① E：②

试题 3(1999 年试题 2)

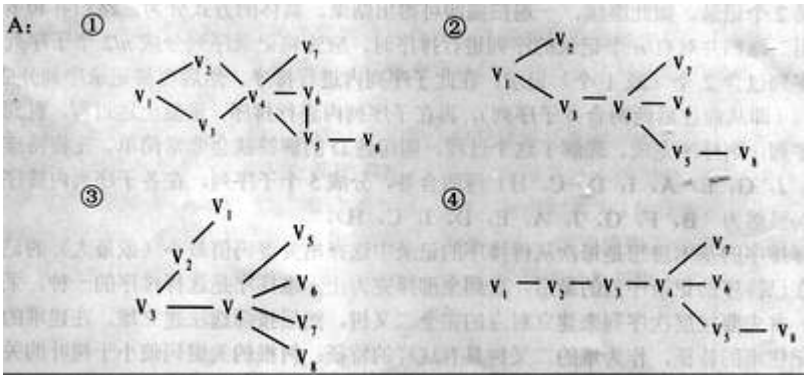
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

给定数据结构(V，E)，V 为结点的有限集合，V={V1，V2，V3，V4，V5，V6，V7，V8}，E 是 V
上关系的集合。

E={＜V1，V2＞，＜V3，V4＞，＜V5，V8＞，＜V6，＜V1，V3＞，V4＜，V5＞，V6＜，＜V1
，V3＞，V4＜，V5＞，＜V2，V4＞，V4＜，V6＞＝它所对应的图表是{ A }，这就是{ B }。

图的存储结构主要有邻接表和{ C }，若用邻接表来存储一个图，则需要保存一个{ D }存储的结点表和若干个{ E }存储的关系(又称边表)。

供选择的答案

A： 

B：①树 ②无向图 ③有向图 ④无回路图

C：①转称矩阵 ②邻接矩阵 ③状态矩阵 ④优先矩阵

D：①顺序 ②链接 ③散列 ④分块

【解析】

本题是一道关于图的试题，应该说是比较简单的。

在数据结构(V，E)中，有关系＜V1，V2＞和＜V1，V3＞，这样很快就答案案③和④排除了，
然后再看答案①②。在数据结构(V，E)中，还有关系＜V2，V4＞，这在答案②中没有反映出
来，所以又排除了答案②。既然确定了答案①就是正确的答案，仔细观察该图形，由于关系
＜V1，V2＞、＜V1，V3＞、＜V2，V4＞，＜V3，V4＞以及关系＜V4，V6＞、＜V6，V5＞、＜
V4，V5＞的特殊性，我们排除了该图是树、有向图、无回路图的可能性，数据结构(V，E)是
图。

图的存储结构主要有邻接表和邻接矩两种。邻接矩阵是表示结点间的相邻关系的矩阵，若 G
是一个具有 n 个结点的图，则 G 的临界阵是如下定义的 n×n 矩阵：

A［ij］=0，表示(Vi，Vj)或者(Vj，Vi)是 G 的边；

A［ij］=1，表示(Vi，Vj)或者(Vj，Vi)不是 G 的边；

用邻接表来存储一个图，需要保存一个顺序存储的结点表和 n 个链接存储的边表。结点表的
每个表目对应于图的一个结点，每个表目包括两个字段：一个是结点的数据或指向结点数据

的指针，另一个是指向此结点的边表的指针。图的每一个结点都有一个边表，一个结点的边
表的每个表目对应于与该结点相关联的一条边。它的每个表目也包括两个字段：一个是与此
相边相关联的另一个结点的序号，另一个是指向边表的下一个表目的指针。对于有向图来说
，用邻接表保存该图可以保存每个结点的出边表或入边表。

【答案】：A：① B：② C：② D：① E：②

试题 4(1998 年试题 3)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的答案，把相应编号写在人卷的对应栏内。

在内部排序中，通常要对被排序数据序列进行多趟扫描。各种排序方法有其不同的排序实施
过程和(时间)复杂性。

对给定的整数序列(541，132，984，746，518，181，946，314，205，827)进行从小到大的
排序时，采用冒泡排序和直接选择排序时，若先选出大元素，则第一趟扫描结果分别是{ A }和{ B }；采用快速排序(以中间元素 518 为基准)的第一趟扫描结果是{ C }。

设被排序数据序列有 n 个元素，冒泡排序和直接选择排序的复杂性是{ D }；快速排序的复杂性是{ E }。

供选择的答案

A～C：①(181，132，314，205，541，518，946，827，746，984)
②(541，132，827，746，518，181，946，314，205，984)
③(205，132，314，181，518，746，946，984，541，827)
④(541，132，984，746，827，181，946，314，205，518)
⑤(132，541，746，518，181，946，314，205，827，984)
⑥(132，541，746，984，181，518，314，946，205，827)

D、E：①$O(n\log_2 n)$ ②$O(n)$ ③$(nccn)$ ④$O(n^2)$
⑤$O(\log_2 n)2$ ⑥$O(n^2\log_2 n)$

【解析】

冒泡排序：冒泡排序的过程很简单。首先将第 1 个数与第 2 个数相比较，若为逆序则交换两数
，然后比较每两个数与第 3 个数，依次类推，直到第 n-1 个数与第 n 个数进行过

比较为止。上
述过程称为一趟冒泡排序，结果是最大的数被称鲐了最后。然后进行第 2 趟，对前面 n-1 个数
进行冒泡排序，结果是次大的数被移到了 n-1 的位置上。一般来说，第 i 趟冒泡排序是从第 1
个数到第 n-i+1 的位置上，整个排序过程需进行 k(1≤k≤n)趟。

分析冒泡排序的效率，若初始序列为正序，则只进行一次排序。在排序过程中只进行 n-1 次
比较，不交换数据。若为逆序，则需进行 n-1 趟排序，需进行 n(n-1)/2 次比较，交换数据的数量组也相同。因此，冒泡排序的复杂性是 $O(n^2)$。

快速排序是对冒泡排序的一种改进，其基本思想是通过一趟排序将待排序的数据分成两部分
，其中一部分的关键字均比另一部分的关键字小，然后再对这两部分分别进行快速排序，最
后达到整个序列有序。快速排序的复杂是 $O(n\log_2 n)$。

直接选择排序，又称简单选择排序，其基本思想是每一趟在 n-i+1(i=1，2，…，n-1)个数据
中选择最小的数据作为有序序列中的第 i 个数据。一趟直接选择排序的基本操作为通过 n-i 次
关键字的比较，从 n-i+1 个数据中选出关键字最小的数据，并和第 i 个数据交换。直接选择排
序过程中，所需交换数据的次数较少，最小值为"0"，最大值为 3(n-1)"。然而，无论数
据的初始次序如何，它所需进行的关键字的比较次数相同，均为 n(n-1)/2，因此，直接选择
排序的复杂性是 $O(n^2)$。

对于题中给定的整数序列(541，132，984，746，518，181，946，314，205，827)进行从小
到大排序，若先选出较大的元素，则对于冒泡排序，第 1 趟操作为 541←→132，984←→746
，984←→518，984←→181，984←→946，984←→314，984←→205，984←→827，其结果
得到的序列为(132，541，746，518，181，946，314，205，827，984)；对于直接选择排序
，第 1 趟操作为 984←→827，其结果得到的序列为(541，132，827，746，518，181，946，3
14，205，984)。

采用快速排序(以中间元素 518 为基准)的第 1 趟扫描结果是(205,132,314,181,518，746
，946，984，827)。

【答案】：A：⑤ B：② C：③ D：④ E：①

试题 5(1997 年试题 4)

从供选择的答案中，选出应填入下面的叙述中{ }内的最确切的解答，把相应编号写在答卷
的对应栏内。

设数据结构(D，R)由数据结点集合 D={di|1≤i≤7}及其上的关系 R 组成。

1.当 R={＜d_{i-1}，d|d_{i-1}，di∈D，2≤i≤7}，这个数据结构对应于 A。

2.当 R={＜d4，d2＞，d1＞，＜d2，d3＞，＜d4，d6＞，＜d6，d5＞，＜d6，d7＞}，这个结构的图形是 B，用 C 遍历法可以得到 A 的数据结构。

3.当 R={＜d1，d2＞，＜d1，d3＞，＜d2，d4＞，＜d4，d5＞，＜d4，d5＞，＜d4，d6＞，
＜d4，d7＞}，这个结构的图形是 D。用 E 遍历法可以得到 A 的数据结构。

供选择的答案

A、B、D：①二叉树 ②队列 ③二叉排列序树
④线性表 ⑤无向图 ⑥有向无回路

C、E：①前序 ②中序 ③后序
④深度优先 ⑤广度优先

【解析】

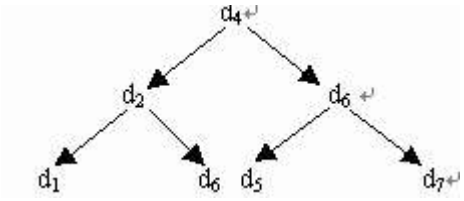
该题要求考生熟练掌握数据结构中常用的线性表、二叉树、树、图的结构及其遍历方法。数
据结构(D，R)中，D 由 7 个数据组成，R 决定这 7 个数据的排列方式。

(1)R={＜d_{i-1}，d|d_{i-1}，di∈D，2≤i≤7 }

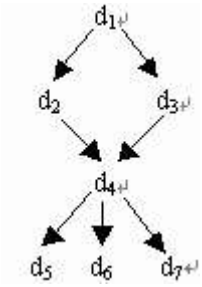
其对应的数据结构可以表示为 d1→d2→d3→d4→d5→d6→d7，它对应一个线性表。

(2)R={＜d4，d2＞，＜d2，d1＞，＜d2，d3＞，＜d4，d6＞，＜d6，d5＞，＜d6，d7＞}

其对应的数据结构可以表示为：



这棵二叉树利用中序遍历法得到序列为 d1d2d3d4d5d6d7。



(3)R={＜d1，d2＞，＜d1，d3＞，＜d2，d4＞，＜d3，d4＞，＜d4，d5＞，＜d4，d6＞，＜
d4，d7＞}，对应的数据结构为一个有向无回路图，利用广度优先遍历法得到的序列为 d1d2d
3d4d5d6d7，正好对应(1)中的线性表结构。

【答案】：A④ B① C② D⑥ E⑤

试题 6(1996 年试题 1)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的
对应栏内。

一棵二叉排序树可顺序存放在一组物理上相邻的存储区中，每个结点及左、右针依次分别放
在该存储区的 3 个连续单元中。现对一棵结点按字母的字典顺序构成的二叉排序树从根结点 P
开始顺序放在一个存储区中，结果如图 2-2 所示。其中 Li 为第 i 个结点的左指针，R 为第 i 个结点的右指针，则 L2 应为 A，L4 应为 B，R1 应为 C。该二叉排序树的前序遍历序列为 D，后序遍历序列为 E。

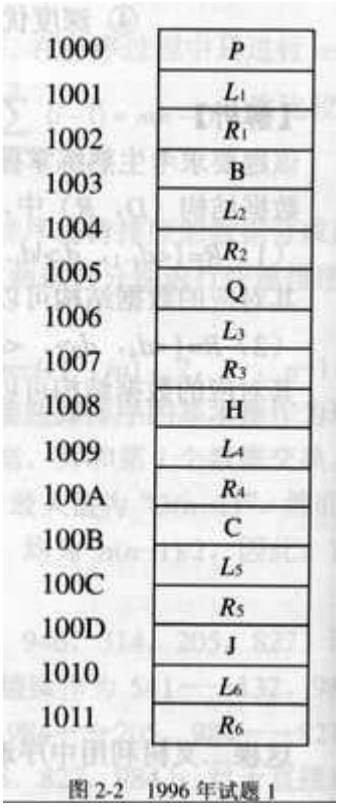


图 2-2 1996 年试题 1

供选择的答案：

A～C：①1003 ②1004
③100A… ④1009
⑤1006 ⑥1000
⑦100C ⑧100F
⑨Null

D、E：①PBQHCJ ②PBHCJQ
③BCHJPQ ④CJHBQP ⑤BHCJQP

【解析】

二叉树或者为空，或者由一个根结点加上左子树和右子树(互不相交的两棵二叉树)构成，因
此，若依次遍历根、左子树、右子树，就有 6 种遍历方法，即 DLR、DRL、LDR、RDL、LRD 和 RL
D。限定先左后右的顺序，则有常用的前序遍历、中序遍历和后序遍历 3 种情况。

基于二叉树的递归定义，可得遍历二叉树的递归算法定义：

(1)先序遍历(DLR)算法：若二叉树为空，则进行的是空操作，否则访问根结点：

前序遍历左子树；

前序遍历右子树；

(2)中序遍历(LDR)算法：若二叉树为空，则进行的是空操作，否则

中序遍历左子树；

访问根结点；

中序遍历右子树。

(3)后序遍历(LRD)算法：若二叉树为空，则进行的是空操作，否则

后序遍历左子树；

后序遍历右子树；

访问根结点。

二叉排序树有如下特点：每个结点的左子树中所有结点的值都小于该结点的值，而右子树中
所有结点的值都大于该结点的值，可知由根结点 P 始，按结点字母的字典顺序构成的二叉树
如图 2-3 所示。

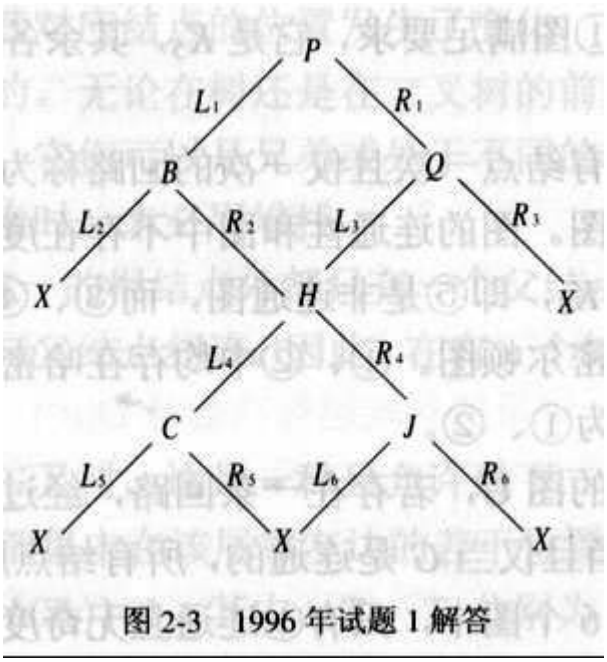


图 2-3 1996 年试题 1 解答

由图 2-3 可得出：L2 指向 Null；L4 指向 C，即 100C；R1 指向 Q，即 1006；该二叉树排序权前序遍历序列为 PBHCJQ，后序遍历序列为 2CJHBQP。

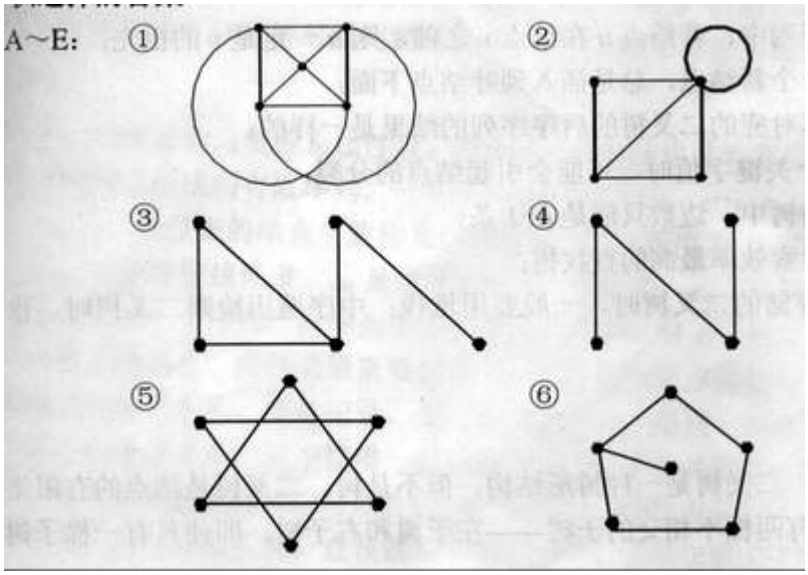
【答案】A：⑨ B：⑦ C：⑤ D：② E：④

试题 7（1996 年试题 14）

从供选择的答案中，选出应填入下面叙述中 内的最确切的解答，把相应编号写在答卷的
对应栏内。

下列图中，A 是非简单图，B 是完全图，C 和 D 都是哈密尔顿图，其中 C 又是欧拉图， E 是树。

供选择的答案：



非简单图：既无平先边也无环的无向图称为简单无向图，有平先边或有环的图是非简单图。
在 6 个图中只有②有环，其余的 5 个图都既无平先边也无环，因而 A 的答案应为②。

完全图：每个顶点度数都等于(n-1)的 n 阶无向简单图称为 n 阶无向完全图，并记作 kn。在给
定的 6 个图中，只有①图满足要求，它是 k5，其余各图均不不完全图，所以 B 的答案为①。

哈密尔顿图：通过图中所有结点一次且仅一次的回路称为该图的哈密尔顿回路，具有哈密尔
顿回路的图称为哈密顿图。图的连通性和图中不存在度数为 1 的顶点都是哈密尔顿图的必要
条件。⑤图是两分离的 K3，即⑤是非连能图，而③、④、⑥已均有度数为 1 的顶点，所以③
、④、⑤、⑥都不是哈密尔顿图。①、②中均存在哈密尔顿回路，因而①、②均

为哈密尔顿
图。所以 C、D 的答案为①、②。

欧拉图：给定无孤立结点的图 G，若存在一条回路，经过图中每边一次且仅一次，该条路称
为欧拉回路。无向图 G，当且仅当 G 是连通的，所有结点度数全为偶数，且有欧拉回路，该无
向图称为无向欧拉图。在 6 个图中，只有①连通且无奇度顶点，因而①为欧拉图，其余各图
中，⑤无奇度顶点，但它不连通，所以不是欧拉图。而②、③、④、⑥中都有奇度顶点，因
而它们也不是欧拉图，所以 C 的答案为①。

树：连通且无回路的无向图称为无向树。6 个图中只有④满足要求，其余 5 个图中均有回路，
因而都不是树，E 的答案为④。

【答案】A：② B：① C：① D：② E：④

试题 8(1995 年试题 3)

从下列有关树的叙述中，选出 5 条正确叙述，并按编号从小到大的次序写在答卷的 A～E 栏内。

①一棵二叉树的层次遍历方法只有前序法和后序法两种；

②在哈夫曼树中，外部结点的个数比内部结点个数多 1；

③完全二叉树一定是平衡二叉树；

④在二叉树的前序序列中，若结点 u 在结点 v 之前，则 u 一定是 v 的祖先；

⑤在查找树中插入一个新结点，总是插入到叶结点下面；

⑥树的后序序列和其对应的二叉树的后序序列的结果是一样的；

⑦对 B 树删除某一个关键字值时，可能会引起结点的分裂；

⑧在含有 n 个结点的树中，边数只能是 n-1 条；

⑨最佳查找树就是检索效率最高的查找树；

⑩中序遍历二叉链存储的二叉树时，一般要用堆栈；中序遍历检索二叉树时，也必须使用堆栈。

【解析】

由二叉树的定义可知，二叉树是一种树形结构，但不是树。二叉树是结点的有限集，可以是空集。任一结点至多有两棵不相交的子树--左子树和右子树。即使只有一棵子树也必须区分清它是左子树还是右子树。

二叉树与树有着密切的关系：任一棵树都可用一棵二叉树唯一地表示。当然树中结点之间的
父子、兄弟的关系在它所对应的二叉树中会有变化。树和二叉树的层次遍历方法有前序法、
后序法、中序法三种。

由于树转换为二叉树时，其对应结点的位置发生了变化，因此树的后序序列和其对应的二叉
树的后序序列一般是不同的。无论在树还是在二叉树的前序序列中，排序在前的结点未必是
排序在后的结点父结点，它们可以是兄弟或处于不同的子树中。二叉树用中序遍历方法存储
时，一般用堆栈，而检索时，未必用堆栈。

在树中，每个结点，除了唯一的根结点外都只有一个父结点，而根结点是没有父结点的。由
于每个结点有且仅有一条边与父结点相连，因此，在有 n 个结点的树中，其边数只能是 n-1 条
。

完全二叉树是一种特殊的二叉树，这是一种只允许最下二层结点的度数小于 2(其他均为 2)，
并且最下面一层的结点都集中在该层靠左边的若干位置上。在二叉树中定义非空二叉树 T 的
高度 h(T)=max{h(T1),h(T2)}+1。其中，T1、T2 分别为 T 的左子树和右子树。若一棵二叉树中
任一结点的左子树高度与右子树高度之差不超过 1，则该称二叉树为平衡二叉树。根据完全
二叉树与平衡树的定义，可知完全二叉树一定是平衡二叉树。

查找树也是一种二叉树，若二叉树 T 的结点按中序遍历，结点 u 的左子树的所有结点的键值都
小于结点 u 的键值,而且结点 u 的右子树的所有结点的键值都大于结点 u 的键值。查找树有 3 种操作：查找、插入和删除。当向查找树插入一个新结点时，总是插在叶结点的下面。

对给定的查找二叉树，在结点的每个空指针上都有附加结点作为该结点的子结点，称这些附
加结点为外部结点,而原树的结点称为内部结点。哈夫曼树就是一种扩充二叉树。

在查找二叉树中查找结点，有许多种方法。最大查找时间在 O(log2n)和 O(n)之间。由于所查
找的结点是随机的，因此将平均查找时间作为代价函数。最佳查找树就是通过构造出各种可
能的查找树，计算出每棵查找树的代价，然后挑选其中代价最小的树而获得的，因而检索效
率是最高的。

B 树是一种平衡的多叉树，是索引文的有效结构。在 B 树中删除某一键值时，常采用与分裂相
反的处理过程--联接，这样做有可能使整棵 B 树减少一层，但不会引起结点的分裂。

【答案】：A：② B：③ C：⑤ D：⑧ E：⑨

试题 9(1994 年试题 15)

从供选择的答案中，选出应填入下面有关排序算法复杂性的叙述中{ }内的正确答案，把编
号写在答卷的对应栏内。

对由 n 个记录所组成的有按关键码排序时，下列各常用排序算法的平均比较次数分别是：二
路归并排序为 A，冒泡排序 B，快速排序为 C。其中，归并排序和快速排序所需要的辅助存储
分别是 D 和 E。

供选择的答案
A-E：①0(1) ②(nlog2n) ③0(n) ④0(n^2)
⑤0（n(log2n)2）⑥(nlog2n)

【解析】

排序就是对由 n 个记录所组成的表按某个关键字的值的大小对记录重新排列。这是数据处理
中常见的重要运算。排序的方法很多，衡量一个排序方法的优劣，主要看排序算法的时间复
杂性和空间复杂性，即平均运算次数及所需要的辅助存储空间的大小。

冒泡排序(bubble sort)：从上到下对每对相邻记录比较关键字大小，使较小关键字的记录
上升，比较一遍后最大的关键字的记录将排在最后，再对其余记录重复上述方法，反复执行
，直到无记录上升为止，n 个记录的平均运算次数是 O(n)2。

归并排序(merge sort)：是把待排序的文件分成 n 个已排序的子文件，将这些文件合并得到
完全排序的文件。n 个记录的平均运算次数是 O(nlog₂n)，所需的辅助存储空间是 O(n)。

快速排序(quick sort)：又称分区交换排序。先取一个记录作为基准，用交换的办法把所有
记录按关键字值比基准值大或小分列在基准记录的两边，然后再分别对这两边的记录重复上
述步骤，直到排序完成。n 个记录的平均运算次数是 O(nlog₂n)，所需的辅助存储空间是 O(log₂n)。

【答案】：A：② B：④ C：② D：③ E：⑥

试题 10(1993 年试题 6)

从供选择的答案中，选出应填入下面关于数据结构叙述中{ }内的正确答案，把编号写在答
卷的对应栏内。

图 2-4 所示是带权的有向图 G 的邻表表示法。以结点 V1 出发，深度遍力图 G 所得的结点序列为 A；广度遍力图 G 所得的结点序列是 B；G 的一个拓扑序列是 C；从结点到 V1 到结点 V8 的最短路径是 D；从结点 V1 到结点 V8 的关键路径是 E。

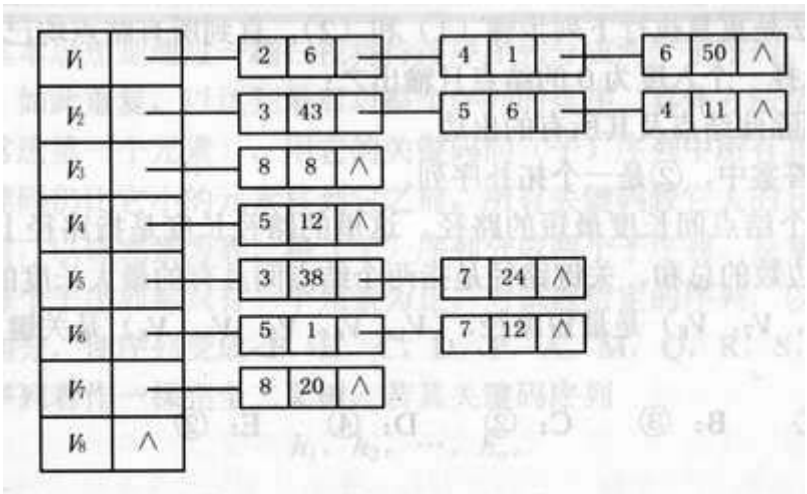


图 2-4 1993 年试题 6

供选择的答案

A～C：①V1，V2，V3，V4，V5，V6，V7，V8
②V，V2，V4，V6，V5，V3，V7，V8
③V1，V2，V4，V6，V3，V5，V7，V8

④V1，V2，V4，V6，V7，V3，V5，V8
⑤V1，V2，V3，V8，V4，V5，V6，V7
⑥V1，V2，V3，V8，V4，V5，V7，V6
⑦V1，V2，V3，V8，V5，V7，V4，V6

D、E①(V1，V2，V4，V5，V3，V8)
②(V1，V6，V5，V3，V8)
③(V1，V6，V7，V8)
④(V1，V2，V5，V7，V8)

【解析】

邻接表表示法可保存一个顺序存储的结点表，是图常用的一种存储方式。结点表的每一个表
目对应于图的一个结点，包括结点的数据或指向结点数据的指针，以及指向此结点的边表的
指针的两个字段。边表的每个表目对应于与该结点相关联的一条边，结点表的每个表目对应
一个链接存储的边表，包括与该结点相关联的一个结点的序号及指向边表下一个表目的指针
的两个字段。因此，题中邻接表所对应的如图 2-5 所示。

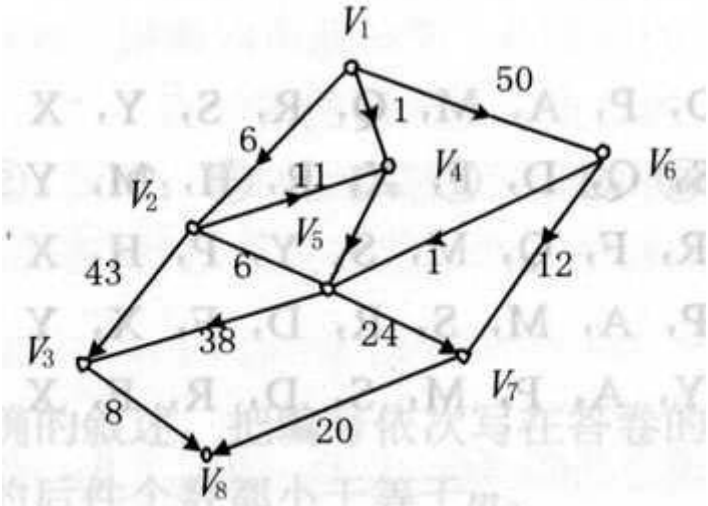


图 2-5 1993 年试题 6

图的深度优先遍历是从图中某个结点 V1 了发，访问此结点，然后依次从 V1 的未被访问的相邻
结点出发进行深度优先遍历，直至图中所有和 V1 有路径相通的结点都被访问到。此时图中若

尚有未被访问的结点，则另选图中一个未被访问到的结点作起始点。重复上述过程，直至图
中所有结点都被访问到为止。因此，邻接表图的深度优先遍历是 V1，V2，V3，V8，V5，V7，
V4，V6。

广度优先遍历是先访问结点 V1，然后访问 V1 连接到的所有未被访问的结点 V2，V3，…，Vt，
再依次访问 V2，V3，…，Vt 连接到的所有未被访问的结点。如此进行下去，直到访问遍所有
结点，因此邻接表图的广度优先遍历是 V1，V2，V4，V6，V3，V5，V7，V8。

如果有向图的某个结点序列满足如下条件：若从结点 Vi 到 Vj 有一条路径，则在序列中结点 Vi
必定在 Vj 之前，则称该序列是一个拓扑序列。任何无环有向图的结点都可以排在一个拓扑序
列中。

拓扑排序的方法是重复执行下列步骤(1)和(2)，直到所有结点均已被输出。

(1)从图中选择一个入度为 0 的结点且输出之；

(2)从图中删除此结点及其所有的出边。

在可供选择的答案中，②是一个拓扑序列。

最短路径是两个结点间长度最短的路径。这里的路径长度是指路径上的边所带权的总和，而
不是路径上边数的总和。关键路径是指两个结点间具有的最大长度的路径。通过计算可知，
(V1，V2，V5，V7，V8)是最短路径，(V1，V6，V5，V3，V8)是关键路径。

【答案】A：⑦ B：③ C：② D：④ E：②

试题 11(1992 年试题 6)

从供选择的答案中选出应填入{ }内的正确答案，把编号写在答卷的对应栏内。

在内排序的过程中，通常需要对待排序的关键码集合进行多遍扫描。采用不同排序方法，会
产生不同的排序中间结果。设要将序列＜Q，H，C，Y，P，A，M，S，R，F，X＞中的关键码按字母的升序重新排列，则 A 是冒泡排序一趟扫描的结果，B 是初始步长为 4 的希尔(Shell)排序一趟扫描的结果，C 是两路归并(合并)排序一趟扫

描的结果，D 是以第一个元素为分界元素的快速排序一趟扫描的结果，E 是堆排序初始建堆的结果。

供选择的答案

A～E：①F，H，C，D，P，A，M，Q，R，S，Y，X
②P，A，C，S，Q，D，F，X，R，H，M，Y
③A，D，C，R，F，Q，M，S，Y，P，H，X
④H，C，Q，P，A，M，S，R，D，F，X，Y
⑤H，Q，C，Y，A，P，M，S，D，R，F，X

【解析】

冒泡排序将待排序的记录顺次两两比较，如为逆序则进行交换，将待排序序列依此法从头至尾处理一遍称作一趟冒泡，一趟冒泡的效果是将关键码值最大的元素交换到了最后位置。对于试题给定的序列，经一趟冒泡后变成序列＜H，C，Q，P，A，M，S，R，D，F，X，Y＞。

希尔排序也是一种插入排序，其特点是把待排序序列每隔某个"步长"的元素组成一个子序
列，分别对每个子序列进行直接插入排序，排序后的结果称为一趟扫视。每趟减少相隔步长
，直至步长为 1，对整个序列作一次扫视。对于试题给定的序列，取初始步为 4，希尔排序一
趟扫视后，将序列变成为＜P，A，C，S，Q，D，F，X，R，H，M，Y＞。

两路归并排序是一种最简单的归并排序，它将含 n 个元素的待排序序列看作 n 个已排序的子序
列。首先将每两个子序列归并，得到 n/2 个已排序的较大的子序列；再对这些子序列归并，
如此反复，直到最后归并成一个序列，排序即告完成。由上述两路归并的方法可知，对于试
题所给的序列，经两路归并排序的一趟扫描后，序列变成＜H，Q，C，Y，A，P，M，S，D，R
，F，X＞。

快速排序的基本思想是通过一趟扫视将待排序序列分成两个子序列，然后分别对这两个子序
列进行排序，如此重复，以达到最后对整个序列的排序。具体方法是：任选待排序序列中的
某元素(通常选第一个元素)，用它的关键码同(子)序列中所有其余元素的关键码相比较，将
所有关键码仍比它小的元素移到它之前，所有关键码较它大的元素都移到它之后。经这样一
趟扫视后，以该元素为界，将(子)序列分成两个子序列。反复对上述子序列作同

样的操作，
直至每个子序列都只有一个元素为止。对试题给定的序列，以它的第一个元素为分界元素作
一次划分，使序列都只有一个元素为止。对试题给定的序列，以它的第一个元素为分界元素
作一次划分，使序列变成＜F，H，C，D，P，A，M，Q，R，S，Y，X＞。

堆将待排序序列看作一棵完全二叉树，若其关键码序列

h1,h2,…，hn.

满足以下性质：

hi≤h2i，hi≤h_{2i+1}，(i=1，2，…，［n/2］)

则称该序列是一个堆。即子树根结点 hi，其关键码值比它的左、右子结点
h2i(2i≤n)和 h_{2i+1}(2i+1≤n)的关键码值都要小。

堆排序的基本思想如下：由于堆的根结点的关键码值是最小的，取走根结点，让剩下的其余
结点重新建成堆，则能从新堆的根结点取走次最小的。如此反复，得到一个递增的排序序列
。堆排序的基本问题是如何建堆，建立初始堆的方法是首先将第 1 个至第 n 个结点分别看作是
只有一个结点的堆，然后逐步把结点第［n-1］个、第［n/2］-1 个…第 1 个结点，分别加到
它们的两棵左、右子树上构成堆，最终变成一个堆。

对于试题给定的序列，经上述初始建堆步骤后，序列变成＜A，D，C，R，F，Q，M，S，Y，P
，H，X＞。

【答案】A：④ B：② C：⑤ D：① E：③

试题 12(1991 年试题 3)

从下列叙述中选出 5 条正确的叙述，把编号依次写在答卷的 A～E 栏内。

①m 阶 B-树每一个结点的后件个数都小于等于 m。

②m 阶 B-树每一个结点的后件个数都大于等于［m/2］。

③m 阶 B-树具有 k 个后件的非叶子结点含有 k-1 个键值。

④m 阶 B-树的任何一个结点左右子树的高度都相等。

⑤中序遍历一棵查找树的结点就可得到排好序的结点序列。

⑥用指针的方式存储一棵有 n 个结点的二叉树，最少要 n+1 个指针。

⑦任一查找树的平均查找时间都小于用顺序查找法查找同样结点的线性表的平均查找时间。

⑧平衡树一定是丰满树。

⑨已知树的前序遍历并不能唯一地确定这棵树，因为不知道树的根结点是哪一个。

⑩不使用递归，也可实现二叉树的前序、中序及后序遍历。

【解析】

m 阶 B-树是一种平衡的 m 叉树，具有如下的性质：

(1)每个结点的后件个数小于等于 m;

(2)除了根结点和叶结点之外，每个结点的后件个数大于等于(m/2)；

(3)具有 k 个后件的非叶结点含有 k-1 个键值;

(4)所有叶结点在同一层上，而且不附有信息。

m 阶 B-树是平衡树，其上任一结点的所有子树的高度都是相等的。

具有下列性质的二叉树称为查找树：除叶结点外，每个结点的键值大于其左子树上一切结点
的键值，且小于等于其右子树上一切结点的键值。

中序遍历法访问二叉树结点的过程是：递归地访问左子树，然后访问树根，再访问右子树。
因此，用中序遍历法遍历一棵二叉查找树可以得到按键值升序排列的结点序列。在用链接表
表示法存储 n 个结点的二叉树上，最多有 2n 个指针，其中 n+1 个是空指针，n-1 个指针用于指
向后件。因此，最少要 n+1 个指针的说法是不对的。

在给定的查找树上，由根结点到所有其他结点的路径长度总和称为内部路径长度。当一棵二
叉树退化为线性链表时，它的内部路径长度最大，等于 n(n-1)/2，所以退化二叉树的查找时
间(包括平均查找时间)等于线性表顺序查找时间。答案⑦是错误的。

具有最小内部路径长度的树是丰满树，对丰满树的查找树进行插入或者删除操作后，会产生
一棵非丰满树。平衡二叉树是指其上任一结点的左右子树的高度(或者结点个数)保持一定比
例的树，即平衡树上任一结点的左、右子树仍然保持平衡。平衡树的查找效率和丰满树相近
，但是在插入或者删除结点时容易产生仍保持平衡的树。由丰满树同平衡树定义可知，丰满
树一定是平衡树，但是平衡树不一定是丰满树。

用前序遍历法访问一棵树的原则是：递归地访问树根，然后访问左子树，再访问右子树。因
此，前序遍历第一个被访问的结点就是原树的根结点，"不知道树的根结点"的说法是错误
的。二叉树的前序、中序和后序遍历法都是递归的。因此，最适合使用递归程序方法实现这
3 种遍历算法。当然不用递归程序方法也能够实现这 3 种遍历算法，但是程序比较复杂，一般
需要用栈或队列操作来实现。

【答案】A：① B：③ C：④ D：⑤ E⑩

试题 13(1990 年试题 4)

从供选择的答案中选出应填入下列叙述中的{ }内的正确答案，把编号写在答案的对应栏内。

在查找算法，可用平均查找长度(记为 ASL)来衡量一个查找算法的优劣，其定义为：

$$ASL = \sum_{i=1}^{n} PiCi$$

此处 Pi 为表中第 i 个记录被查找的概率，Ci 为查找第 i 个记录时同关键字比较的次数，n 为表
中表有记录数。

以下叙述中均假定每一个记录被查找的概率相等，即 Pi=1/n(i=1，2，…，n)。

当表中的记录连续存储备在一个一维数组中时，可采用顺序查找与折半查找方法(折半查找
要求表是按关键字有序排列的)。顺序查找时的 ASL 为 A，折半查找时的 ASL 为 B。记录的关键

字有序时，用二叉排序树查找记录，在最坏的情况下，ASL 为 C。当二叉排序树是一棵平衡树
时，ASL 为 D。在平衡树上删除一个结点后可以通过旋转使其平衡，最坏的情形下需 E 次旋转
。

供选择的答案

A～E：①0(1) ②0(lon^{2}n) ③0(lon^{2}n)2 ④0(nlog$_2$n)
⑤0(n) ⑥0(n^2)

【解析】

顺序查找：比较一次可查到第 1 个元素，比较两次可查到第 2 个元素。一般地，比较 i 次可查
到第 i 个元素。

顺序查找的 ASL 为：ASL=（1/n）*(n(n+1)/2) =n(n+1)/2=0(n)

折半查找：比较一次可找到一个元素；比较两次可找到两个元素；一般地，比较 i 次可找到 2
i-1 个元素。总共 n 个元素，最多需比较 log$_2$(n+1)次。

折半查找的 ASL 为：ASL

$$ASL=\sum_{i=1}^{\log_2(n+1)}\frac{1}{n}\cdot i\cdot 2^{i-1}$$

而归纳法不难证明：

$$\sum_{i=1}^{k} i\cdot 2^{i-1}=2^k(k-1)+1$$

将此式入上式，折半查找的 ASL 为：

ASL=1/n(n+1)(log$_2$(n+1)+1)=1/n(n+1)log$_2$(n+1)-1

=0(log$_2$n)

对于二叉排序树的查找，最坏情况是二叉树中没有结点，有两棵子树。在这种情况下，二叉
排序树查找相当于顺序查找，即最坏情况下，ASLA 为 ASL=0(n)。

对于平衡的二叉排序树查找，平衡二叉排序树中的每个结点的左右子树高度差不大于 1，n 个
结点的平衡树的深度和 log$_2$n 是同数量级。对二叉排序树来说，它的查找时间与

树的深度成
正比。因此，平衡二叉排序树的 ASL 为 ASL=O($\log_2 n$)。

在平衡树上删除或插入结点往往会破坏树的平衡。通过旋转又可使其重新成为一棵平衡树。
在最坏情况下，从叶子开始一直调整到根为止，需进行的旋转次数与树的深度一致，等于 O(
$\log_2 n$)次。

【答案】：A：⑤ B：② C：⑤ D：② E②

试题 14 (1990 年试题 6)
从供选择的答案中选出应填入下列叙述中的{ }内的正确答案，把编号写在答卷的对应栏内。

设 T 是正则二叉树，它具有 6 片树叶，那么树 T 的高度最多可以是 A；最小可以是 B；树 T 的内结点数是 C；如果 T 又是哈夫曼(Huffman)最优树，且各片树叶的权分别是 1、2、3、4、5、6，
则最优树 T 的非树叶的权之和是 D；权为 1 的树叶的高度是 E。(注：树的根结点高度为 1)

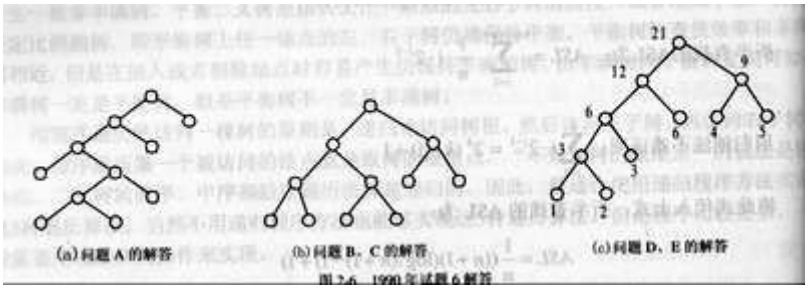
供选择的答案：

A、B、C、E：①7 ②5 ③6 ④4 ⑤3

D：①27 ②30 ③45 ④51 ⑤64

【解析】

这里要注意正则树的概念，所谓正则二叉树，就是每个非叶结点都有两个子结点。根据这个
原则，我们用下面的 2-6(a)、2-6(b)、2-6(c)图来解答本题。



【答案】：A：③ B：④ C：④ D：④ E②

2.2 程序语言基础知识

2.2.1 主要知识点

了解程序语言的种类、特点和适用范围，掌握汇编、编译、解释系统的基本原理，以及程序语言的数据结构和控制结构。特别是编译系统形式的基础知识，应当下功夫掌握。

2.2.1.1 程序语言概述

低级语言又称面向机器语言，它是特定的计算机系统所固有的语言。用机器语言编制出来的,程序可读性很差，程序员难以修改和维护，于是人们考虑改用助记符号来表示机器指令中操作码和操作数，这就是汇编语言。汇编语言仍然是一种和计算机的机器语言十分接近的语言,它的书写格式在很大程度上取决于特定计算机的机器指令。它是一种低级语言，这对于人们抽象思维和交流十分不便。在这个基础上，高级语言就发展起来了。目前已有许多流行较广的高级语言，如Fortra、Cobol、PascalC、和C++等。这类语言与人们的自然语言比较接近，大大提高了程序设计的效率，便于人们进行交流。

计算机只能理解和执行机器语言。程序语言要在计算机上运行,必须有一个程序,使机器能够理解用程序语言书写的用户程序，这就是所谓的语言处理程序。它可以分为两大类：解释程序和翻译程序。

2.2.1.2 程序语言基础知识

程序语言种类繁多，在应用上各有不同的侧重面。

Fortran 是第一个被广泛用于科学计算的高级语言。

Algol 是另一个早期研制出来的高级语言。它有严格的文法规则，用巴科斯范式 BNF 来描述语言的文法。Algol 是一个分程序结构的语言。

Cobal 是一种面向事务处理的高级语言。

Pascal 语言提供的为数不多而又相当紧凑的机制使得这个语言具有相当强的表达能力。

C 是一种通用程序设计语言。C 作为一种较低级的语言，提供了指针和地址操作的功能。C 提供书写结构良好的程序所需的控制结构。C 与 UNIX 操作系统紧密相关，UNIX 操作系统及其上的许多软件都是 C 编写的。

还有几种特殊而又重要的程序语言：

(1)面向对象的 C++
C++是在 C 语言的基础上发展起来的与 C 兼容的语言。主要增加了类功能，成为面向对象的程序设计语言。面向对象程序语言至少包含以下几个重要概念：

①对象是人们要进行研究的任何事物，它包括状态(用数据来描述)和操作(用来改变对象的状态)两方面。面对对象语言把状态和操作封装于对象体之中，并提供一种访问机制，使对象状态的具体表示和操作的具体实现都是隐蔽的。

②类是面向对象语言必需提供的用户定义的数据的类型，它将具有相同状态、操作和访问机制的多个对象抽象成一个对象类。在定义了类以后，属于这种类的一个对象叫作类实例或类对象。

③继承是面向对象语言的另一个必备要素。类与类之间可以组成继承层次，一个类的定义(称为子类)可以定义在另一个已定义类(称为父类)的基础上。子类可以继在父类中的属性和
操作，也可以定义自己的属性和操作。

(2)逻辑型语言 Prolog

逻辑型语言是一类以形式逻辑为基础的语言。Prolog 是这类语言的代表。Prolog 建立在关系理论和一阶谓词理论基础上，具有和传统的命令型程序设计完全不同的风格。Prolog 程序由一些称为事实和规则的 Horn 子句组成。具有很强的推理功能，适用于书写自动定理证明、专家系统、自然语言理解等问题的程序。

(3)函数型语言 LISP
函数型程序语言是一类以 λ 演算为基础的语言。LISP 是典型的函数型程序语言。函数是一种对应规则(映射)，它使其定义域中每一个值和值域中唯一的值相对应。

函数型程序设计语言的优点之一是对表达式中出现的任何函数都可以用其他函数来代替，只要这些函数调用产生相同的值。由于用函数程序设计语言书写的程序是利用自变量的值来计算函数的值的，它没有副作用。这些特点有助于程序模块化的实现。

2.2.1.3 程序语言的数据类型

不同程序语言所提供的数据类型不尽相同。数据是程序操作的对象，具有名称、类型、存储类、作用域和生存期等属性，使用时要为它分配内存空间。数据名称由用户可通过标识符命名：类型说明数据占用内存的大小和存放形式，存储类说明数据在内存中的位置和生存期；作用域说明数据可以使用的范围；生存期说明数据占用内存的时间。

数据从不同角度可分成不同的类别。按数据的作用域大小，可分为全局量和局部量；按生存期可分为自动生存期、静态生存期和动态生成期；按程序运行时数据的值是否能改变可分为常量和变量。

数据按类型可分为 4 种：Void、标量(Scalar)、函数和聚合(Aggregate)。标量又可分为算术、枚举和指针；聚合类型可分为数组、结构体和共用体。
数据按其构造方式又可分为基本类型和派生类型。

2.2.1.4 程序语言的控制结构

程序语言中控制结构为将数据和数据上的运算组合成程序提供了基本框架。可计算问题的程序都可用顺序、选择和循环这 3 种控制结构来描述。

2.2.1.5 汇编程序基础知识

汇编语言是为特定的计算机中计算机系统设计的面向机器的语言。汇编语言中的语句可分成两大类：一类是与机器指令相对应的可执行汇编语句；另一类汇编语句称为汇编控制语句(亦称伪指令)。用汇编语言编写的源程序，要通过汇编程序将它民机器语言程序，才能被计算机理解执行。经过汇编程序的工作，可执行汇编语句被转化成对应的机器指令；而伪指令并不翻译成机器指令，它们的作用是控制汇编程序工作。伪指令主要用来告诉汇编程序做一些除了翻译机器指令外必须做的工作。

汇编程序的功能是将汇编语言所编写的源程序翻译成由机器指令和其他信息组成的目标程序。汇编程序的基本工作包括两项：一是将每一条可执行汇编语句转换成对应的机器指令；二是处理源程序中出现的伪指令。

2.2.1.6 解释程序基础知识

解释程序是一种语言处理程序，它直接执行源程序或源程序的内部形成。它并不产生目标程序，它是它和编译程序的主要区别。

高级语言实现语言处理 4 种方案：

第 1 种，源程序被直接解释执行。

第 2 种，先将源程序翻译成高级中间代码，然后再扫描高级中间代码，对高级中间代码进行解释执行。

第 3 种，也是一种解释程序的实现方案，与第 2 种方案的解释程序不同点在于，首先将源程序转化成和机器代码十分接近的低级中间代码，然后再解释执行这种低纸中间代码。这类系统具有良好的可移植性。

第 4 种，是普通的编译程序。在编译程序方案下，高级语言编写的源程序被最终翻译成机器语言表示的目标程序。这类系统的目标执行效率最高。

一般说来，建立在翻译基础上的系统在执行速度上都优于建立在解释执行基础上的系统。翻译系统的缺点是其复杂性，这使得它的开发和维护费用都大。相反，解释系统比较简单，可称植性较好，适合于以交互方式执行程序，其缺点是执行速度慢。

解释系统的结构可分成两部分。第 1 部分包括通常用的词法分析程序以及语法和语义分析程序，它的作用仍是把源翻译成中间代码，中间代码的设计常采用逆波兰表示形式。第 2 部分是解释部分，用来对第 1 部分所产生的中间代码进行解释执行，完成真正的解释工作。

2.2.1.7 编译程序基础知识

编译程序的功能是把某高级语言书写的源程序翻译成与之等价的低级语言(汇编语言或机器语言)的目标程序。其过程很复杂，可分成 6 个阶段；

①词法分析阶段是编译过程的第 1 个阶段。词法分析所依据的是语言的词法规则，即描述单词结构的规则。词法规则可用 3 型文法(正规文法)或正规式来描述，有限自动机能识别正规文法所定义的语言和正规式所表示的集合。

②语法分析阶段。在词法分析的基础上将单词符合序列分解成各类语法单位。语法分析所依据的是语言的语法规则，即描述程序结构的规则。
词法分析和语法分析本质上都是对源程序的结构进行分析。

③语义分析阶段是审查源程序有无语义错误，为代码生成阶段收集类型信息。比如语义分析的一个工作是进行类型审查，审查每个算符是否具有语言规范允许的运算对象。

④中间代码生成阶段。在进行了上述的语法分析和语义分析阶段工作之后，有的编译程序将源程序变成一种内部表示形式，这种内部表示形式叫做中间语言或中间代码。所谓"中间代码"是一种简单、含义明确的记号系统。

语义分析和中间代码生成所依据的是语言的语义规则。一般采用语法指导翻译规则和中间代码生成规则。

⑤代码优化阶段是对前阶段产生的中间代码进行变换或进行改造，目的是使生成的目标代码更为高级，即省时间和省空间。

⑥目标代码生成阶段。是把中间代码变换成特定机器上的绝对指令代码或可重定位的指令代码或汇编指令代码。这是编译的最后阶段，它的工作与硬件系统的结构和指令的含义有关。

编译过程的 6 个阶段的任务，再加上表格管理和出错处理的工作可分别由几个模块或程序完成，它们分别称作词法分析程序、语法分析程序、语义分析程序、中间代码生成程序、代码优化程序、目标代码生成程序、表格管理程序和出错处理程序。

2.2.2 试题分析

从历年试题统计(见表 2-2)来看，程序语言基础试题是每年必考的知识点，考查的重点是形式语言基础、语法和词法分析方法、程序控制结构。复习中还应注意补充有关程序语言最新发展方面的知识。



高级程序员考试试题分类精解

表 2-2 历年高级程序员级程序语言基础试题统计

试题	考查知识点
1991 年试题 7	编译语法分析器
1992 年试题 2	程序语言常识
1993 年试题 7	文法
1994 年试题 2	子程序调用、程序控制结构（递归和循环）
1994 年试题 16	文法
1995 年试题 15	短语结构文法
1996 年试题 5	有限状态自动机（词法分析）
1997 年试题 1	子程序参数调用
1997 年试题 7	程序语言常识
1998 年试题 4	文法语法分析、算符优先文法
1998 年试题 6	子程序参数调用
1999 年试题 3	文法
2000 年试题 3	文法

试题 1 (2000 年试题 3)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

假设某程序语言的文法如下：

S→a|b|(T)

T→TdS|S

其中：Vt={a,b,d,(,)},Nn={S,T}，S 是开始符号。

考查该文法，称句型(Sd(T)db)是 S 的一个 A。其中 B 是句柄；C 是素短语；D 是该句型的直接短词；E 是短语。

供选择的答案

A：①最左推导 ②最右推导 ③规范推导 ④推导

B：①S ②b ③(T) ④Sd(T)

C：①S ②b ③(T) ④Sd(T)

D：①S ②S，(T)，b
③S，(T)，TdS，b ④(Sd(T)db)

E：①(Sd(T)db) ②d(T) ③Td ④Sd(T)d

【解析】

解答本题要搞清楚基本概念，下面具体分析各个问题。

先来看问题 A。最左(右)推导：任何一步推导过程 σ→β(其中 σ、β 是句型)都是对 σ 中的最左(最右)非终结符进行替换，这种推导为最左(最右)推导。在形式语言中，最右推导常被称为规范推导。

题中的句型(Sd(T)db)的第一步肯定是由 S→(T)→(TdS)得出的。按照最左推导的规则(Tds)→(TdSdS)→(SdSdS)，最终不可能推出原来的句型。

按照最右推导的规则(Tds)→(Tdb)→(Td(T)db)，最终不可能推出原先的句型。

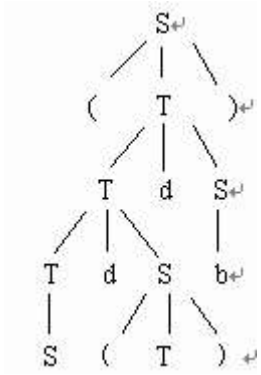
最后可以看出句型(Sd(T)db)是由一般推导推出的，步骤如下：

S→(T)→(Tds)→(Tdb)→(Td(T)db)→(Sd(T)db)

所以正确答案是④。

再来看问题 B～E。来文法，S 是文法的开始符号，α β δ 是文法 G 的一个句型。如果有 S→αAδ，且 A→β，则称 β 是句型 α β δ 相对于非终符 A 的短语。特别是如有 A β，则称 β 是句型 α β δ 相对于规则 A→β 的直接短语。一个句型的最左直接短语称为该句型的句柄。

本文法推导树如下：



所以，S 是句型相对于规则 T→S 的直接短语，也是最左直接短语(句柄)。(T)是句型相对于规则 S→(T)的直接短语，对于问题 B，选择①是正确的。

素短语是一个短语，它至少包含一个终结符，并除自身外不包含其他的素短语。所以，问题 C 的答案③正确。

d 是句型相对于规则 S→d 的直接短语，则问题 D 的答案②正确。

由推导树可知，无论如何，无法由 S 推导出 d(T)、Td 或 Sd(T)d，所以问题 E 的答案①正确。

【答案】： A：④ B：① C：③ D：② E：①

试题 2 (1999 年试题)3

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷扔对应栏内。

假设某程序语言的文法如下：

S→SaT|T

T→TbR|R

R→PdR|P

P→fSg|e

其中 Vr{a,b,d,e,f,g};Vn={S,T,R,P};S 是开始符号，那么，此文法是 A 文法。这种文法的语法分析通常采用优先短阵。优先短阵给出了该文法中各个终结符之间的优先关系(大于、小于、等于和无关系)。在上述文法中，某些终结符之间的优先关系如下：

b{B}a; f{C}g; a{D}a;d{E}d。

供选择的答案

A：①五则文法 ②算符文法 ③二义文法 ④属性文法

B：①大于 ②小于 ③等于 ④无关系

C：①大于 ②小于 ③等于 ④无关系

D：①大于 ②小于 ③等于 ④无关系

E：①大于 ②小于 ③等于 ④无关系

【解析】

所谓算符文法，可以作如下的描述：如果在一个文法 G 中，不含有形如"U→…AB…"的产生式，其中 A,B∈Vn，则 G 为算符文法。也就是说，如果 G 是一算符文法，那么 G 的任何产生式的右部都不会出现两个非终结符号相邻的情况，而且，对算符文法而言，也不会含有两个非终对符号相邻出现的句型。这种性质意味着，如果把终对符号看作广义运算符，而把非终结符号看作广义运算的

对象，则在算符文法的任何句型中，两相邻运算符之间的运算对象至多只有一个，而不会出现其间运算对象个数不确定的情况，这样就使得广义运算总是按照中缀形式出现，对语法分析工作非常有益。

对于给定的文法 G，可以逐个检查 G 的各产生式，查看它们的右部是否含有相邻出现的非终结符号，以确定 G 是否为一算符文法，然后再构造相应的优先矩阵。若此矩阵中无多重定义的元素(即各运算符对之间至多只有一种优先关系)，同则可确认理一算符优先文法。

在算符文法中，一般按照如下规则判断终结符之间的优先关系：

当且仅当 G 中有形如"U→…ab…"或者"U→…aBb…"的产生时，a=b；

当且仅当 G 中有形如"U→…aA…"的产生式，且有或者"$A=^{+}=>b$…"或者"$A=^{+}=>$…aB"时，a＞b；

当且仅当 G 中有形如"U→…Ab…"的产生式，且有或者"$A=^{+}=>a$…"或者"$A=^{+}=>$…aB"时，a＞b。

如果算符文法 G 的任何一对终结符号之间，至多只有 3 种算符优先关系：等于、大于或者小于成立，则称 G 为算符优先文法。

【答案】：A：② B：① C：③ D：① E：②

试题 3 (1998 年试题 4)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

语法分析方法大体上可分成自上而下和自下而上的两种。自下而上分析法，是从输入符号串开始逐步进行 A，直至 A 成文法的起始符号。自上而下分析法，则是从文法的起始符号开始反复使用产生式进行 B 直至 B 出输入符号串。

符符优先文法是一种自下而上分析方法，其文法的特点是文法产生式中 C。自下而下的分析方法，通常要求文法的产生式 D，如 E 文法就是一种可以自上而下分析的文法。

供选择的答案

A、B：①递归 ②综合 ③回归 ④推导 ⑤分解 ⑥归约

C：①不含两个相邻的非终结符 ②不含两个相邻的终结符
③不含 产生式 ④不含长度为 1 的产生式

D：①不以非终结符开头 ②不以终结符开头
③不含左递归 ④不含右递归

E：①LR(1) ②LL(1) ③SLR(1) ④LALR(1)

【解析】

语法分析的程序以词法分析程序所输出的用内部编码表示的单词序列为输入，其任务是分析源程序的结构，差别它是否是相应程序设计语言的一个合法程序。为了完成这个任务，通常由语法分析程序尝试着为其构造一棵完整的语法树。若尝试成功，则表明输入的符号串在结构上是一个合乎语法的程序，否则，源程序中必然存在的错误。

就产生语法树的方向而言，可大致把它们分为自顶向下和自底向上两大类。所谓自顶向下的分析是对给定的符号串，试图自上而下地为其构造出一棵语法树，或者说从文法的开始符号出发，为其构造一个最左推导。所谓自底向上的分析是给定的符号串，试图自下而上地为其构造出一棵语法树，或者说从给定的符号串本身出发，试图将其归约为文法的开始符号。

算符优先文法属于自下而上的分析法，它利用各个算符间的优先关系和结合规则来进行语法分析，特别适用于分析各种表达式。算符优先文法的任何产生式的右部都还会出现两个非终结符相领的情况，且其任何一对终对符之间至多只有 3 种算符关系″＜″、″＞和″=″之一成立。自顶向下的文法中不能有左递归，否则自顶和下的分析过程进入死循环，不能正常进行。E 中 IL(1)为自顶向下的文法外，其他均为自底向上的文法。

【答案】：A：⑥ B：④ C：① D：③ E：②

试题 4 (1998 年试题 6)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的答案，把相应编号写在答卷的对应栏内。

在高级程序设计语言中，使用参数是子程序之间传递信息的一种手段。子程序说明中的参数称为形式参数，调用语句中的参数称为实在参数。调用时，实在参数的个数、类型和顺序要和形式参数保持一致。

知道一种语言(或编译器)使用哪种参数传递方法是很重要的，因为程序的运行依赖于所用的方法。参数传递方法有传值调用(Call by value)、引用调用(Call by reference)、传名调用(Call by name)和宏扩展(Macro expansion)。

传值调用是把实在参数的 A 传递给相应的形式参数，子程序通过这种传值形参 B；引用调用是指把实在参数的 C 传给相应的形式参数，此时子程序对形式参数的一次引用或赋值被处理成对形式参数的 D 访问。

C 语言中的函数，以 E 方式进行参数传递。

供选择的答案

A、C：①地址 ②名 ③值 ④地址和值
⑤值和名 ⑥名和地址

B：①可传回结果的值 ②可传回存放结果的地址
③可传回结果的值和存放结果的地址 ④不可传回任何结果(值或地址)

D：①直接 ②间接 ③变址 ④引用或赋值

E：①传值调用 ②引用调用 ③传名调用 ④宏扩展

【解析】

传值调用是指把实在参数的值传递给相应的形式参数，子程序不能通过这种方式传回任何结果。引用调用是指把实在参数的地址传递给相应的形式参数，此时子程序对形式参数的一次引用或赋值都是对形式参数的间接访问。

C 语言规定，实参变量对形式参数的数据传递是"值传递"，即单向传递，只能由实参传给形参，而不能由形参传给实参。所以说，C 语言中函数是传值调用的。

【答案】A：③ B：④ C：① D：② E：①

试题 5 (1997 年试题 1)

从供选择的答案中，选出应填入下面叙述中 内的最确切的答案，把相应编号写在答卷的对应栏内。

一种最早用于科学计算的程序设计语言是 A；一种提供指针和指针操作且不存在布尔类型的、应用广泛的系统程序设计语言是 B；一种适合在互联网上编写程序可供不同平台上运行的面向对象程序设计语言是 C；一种在解决人工智能问题上使用最多的、有较强的表处理功能的函数程序设计语言是 D；一种以谓词逻辑为基础的、核心是事实、规则和推理机制的实用逻辑程序设计语言 E。

供选择的答案

A～E：①Pascal ②Ada ③Smalltalk ④Snobol ⑤C
⑥Alogo 68 ⑦Java ⑧Lisp ⑨Prologo ⑩Fortran

【解析】

程序设计语言从机器语言、汇编语言到今的高级语言，其发展越来越快，功能越来越强，同时，其可理解性也越来越接爱人类的思维方式。

Fortran 是第一种被用于进行科学计算的高级语言，它出现于 50 年代中期，其设计目的主要暖和于科学计算，它在程序设计语言的发展史上起着突出的作用。

Alogo 68 是另一种早期研制出来的高级语言，虽然没有被广泛使用，但对后来的程序设计语言的发展有着重大影响。

C 语言是目前应用最广泛的系统程序设计语言，它提供了指针与指针操作，且不存在布尔类型，对数据的访问灵活广泛，C 语言还可提供很多底层系统调用，与硬件结合紧密，易用性强。

Java 是适应 Internet 发展的需要而产生的通用网络程序设计语言，它提供了更好的网络安全性和平台无关性，并且采用了面向对象的原理，便于扩展，适合在互联网上编写可供不同平台上运行的面向对象程序设计语言。

Prolog 语言在人工智能领域应用较广泛，是一种以谓词逻辑为基础，借助于推理规则从已有事实推出新的事实的实用的逻辑程序设计语言。

Ada 和 smalltalk 两种语言引入了一定的封装机制，实现了信息隐藏。Ada 的主要封装机制为程序包；Smalltalk 是一种纯面向对象程序设计语言，它侧重于动态链接，不进行任何类型检查，并且类和对象之间没有明显区别。

Snobol 于 1962 年由贝尔实验室开发，它主要的操作为字符串操作。该语言一般不作为通用语言使用，而是作为一种研究工具，用于对文艺作品、音乐等进行分析。

Lisp 是一种在解决人工智能问题上使用最多的、有强的表处理功能的函数程序设计语言。

【答案】A：⑩： B：⑤ C：⑦ D：⑧ E：⑨

试题 6 (1997)年试题 7

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的答案，把相应编号写在答卷的对应栏内。

用高级语言编写程序时，子程序调用语句中的实在参数必须与子程序说明中的形式参数在 A 上保持一致。在允许子程序递归调用的高级语言环境中，需用动态存储管理方法，它通常使用一个 B 存放子程序的调用记录，调用记录可包括(1)全局量存储区域的 C；(2)调用点所在子程序的 D；(3)调用点的 E；(4)形式参数和实在参数的通信区域；(5)返回值；(6)本子程序的局部量和临时变量存储区域等。

供选择的答案

A：①个数、类型 ②个数、顺序
③个数、格式、顺序 ④个数、类型、顺序

B：①线性表 ②队列 ③堆 ④下推栈

C～E：①子程序首地址 ②调用记录首地址
③参数地址 ④寄存器地址
⑤返回地址 ⑥开始地址

【解析】

除少数程序语言之外，形式参数和实在参数之间的对应关系通常按位置来确定。子程序定义中有形式参数表，而子程序调用是用实在参数法表。进入子程序时，第一个形式参数就和第一个实在参数所表示的数据或其他信息相关联，依奖对应。这样，子程序调用语句中的实在参数必须与子程序说明中的形式参数在个数、类型和顺序上保持一致，否则就不能在主程序和子程序之间正确地传递信息。

对于允许子程序递归调用的程序语言，一般采用动态存储管理方法，用下堆栈来实现。由于某一子程序可能被调用了若干次，但只有最近一次调用正处于执行状态，而其余各次调用正等待下次调用的返回。这样，前几次该子程序调用的属于该子程序的局部变量存储区中的内容必须保存起来，以便下次调用返回时再继续使用。子程序执行需要一个用来存放有关信息的区域，这个区域为过程的调用记录，一般包含以下几部分：

- 全局变量存储区域的开始地址；
- 调用点所在子程序的调用记录的地址；
- 调用点机器状态，如返回地址和寄存器当时值；
- 形式参数和实在参数的通信区域；
- 返回值；
- 本子程序的局部量和临时变量存储区。

【答案】：A：④ B：④ C：⑥ D：② E：⑤

试题 7（1996 年试题 5）

从供选择的答案中，选出应填入下面叙述中 内的最确切的解答，把相应编号写在答卷的对应栏内。

有限状态自动机可用五元组(VT，Q， δ，q0，Qf)来描述，它可对应于 A。设有一有限状态自动机 M 的定义如下：

VT={0，1} Q={q0,q1,q2}

δ 定义为：

δ(q0,0)=q1, δ(q1,0)=q2

δ(q2,1)=q2, δ(q2,0)=q2

Qf={q2}

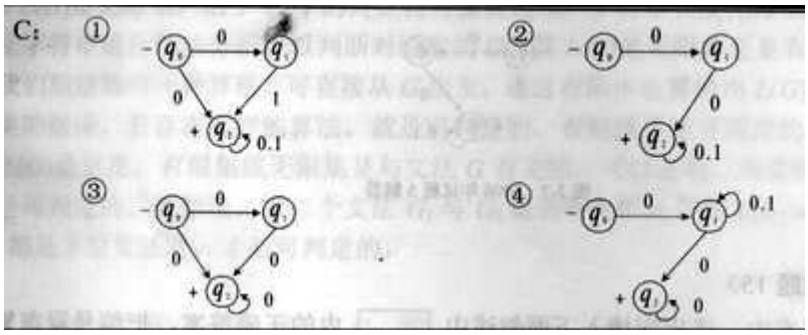
M 是一个 B 有限状态自动机，它所对应的状态转换图为 C，它所能接受的语言可以用正则表达式表示为 D，其含义为 E。

供选择的答案

A：①0 型文法 ②1 型文法 ③2 型文法 ④3 型文法

B：①歧义的 ②非歧义的 ③确定的 ④非确定的

C：



D：①(0|1)* ②00(0|1) ③(0|1)*00 ④0(0|1)*0

E：①由 0 和 1 所组成的符号串的集合。

②以 0 为头符号和尾符号、由 0 和 1 所组成的符号串的集合。

③以两个 0 为结束的、由 0 和 1 所组成的符号串的集合。

④以两个 0 为开始的、由 0 和 1 所组成的符号串的集合。

【解析】

先来看一下地分类表 2-3

表2-3 Chomsky 文法分类表

文法类型	文法名称	语言名称	对应的自动机
0型（PSG）	短语结构文法	递归可枚举语言	图灵机（Turing）
1型（CSG）	前后文相关文法	前后文有关语言	线性限界自动机
2型（CFG）	前后文无关文法	前后文无关语言	非确定下推自动机
3型	正规文法	有限状态语言	有限状态自动机

一个有限状态自动机 M 用 5 元组(VT，Q，δ，q0，Qf)来表示，其中 VT={0，1}是一个有穷字母表，它的每个元素称为一个输入字符：Q=(q0,q1,q2)是一个有限集，它的每个元素称为一个状态；δ 定义为
δ(q0,0)=q1, δ(q1,0)=q2, δ(q2,1)=q2, δ(q2,0)=q2 是一个从 VT×Q 到 Q 上的(单值)部分映射；q0 为初态；Qf={q2}为终态集。

有限自动机分为确定的有限自动机和非确定的有限自动机。确定的有限自动机的确定性表现在映射 δ：VT×Q→Q 是一个单值函数，即对任何状态 q Q 和输入字符 a VT，映射 δ(q,a)唯一确定下一个状态。由分析可知本题给出的是一个确定的有限自动机，它的状态转换如图 2-7 所示。
它所能接受的语言可以用正则表达式表示为 00(0|1)*，其含义为由两个 0 开始的后跟任意个(包含 0 个或多个)0 或 1 组成的符号串的集号。

【答案】：A：④ B：③ C：/ D：/ E：/

(注：原题中 C 的供选择答案有印刷错误，故 C、D、E 的任何解答均作正确处理。)

试题 8 (1995 年试题 15)

从供选择的答案中，选出应填入下面叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

编译技术涉及语言和自动机理论，形式语言可由短语结构文法 G 生成(记为 L(G))，G 可分成 0 型、1 型、2 型、3 型和 4 种类型，与之相对应的自动机分别为图灵机、A、B、C。一个命题的可判定性是指：存在一种算法能给出该命题成立与否的结论。给定文法 G，只有当 G 为 D 时，命题"L(G)是空集，有限集或无限集"才是可判定的，当给出二个不同文法 G1 和 G2，只有当 G1、G2 都是 E 时命题"L(G1)=L(G2)"才是可判定的。

供选择的答案

A～C：①自动机 ②下推自动机 ③双向自动机
④线性界限自动机 ⑤有限状态自动机 ⑥非线性自动机

D～E：①1 型 ②2 型 ③3 型 ④0 型
⑤2 型或 3 型 ⑥1 型或 2 型或 3 型 ⑦0 型或 1 型或 2 型或 3 型

【解析】

用计算机对自然语言进行完全自动处理是一件十分困难的事情。这是因为自然语言歧义性哟，用形式化的语法描述起来很困难。为了便于计算机的自动处理，语言的形式化描述便显得十分重要。现有算法语言在形式上都是形式语言。

形式语言可由短语结构文法来生成。一个短语结构文法 G 是一个 4 元组：G=(VT,VN,P,S)。其中 VT 是终止符的非空有限集，VN 是非终止符的变量的非空有限集，S 为起始符，P 是产生式集。P 中的每个产生式又称重写规则，其一般形式是 α→β，其中 α、β 均为字符串。由起始符开始，不断使用 P 中产生式，可能得到终止符组成的字符串。这种终止符串的全体就是文法 G 生成的语言，记为 L(G)。

按照对产生式中 α 和 β 的不同限制，文法 G 可分成 0 型、1 型、2 型、3 型 4 类，又分别称图灵文法、上下文有关文法、上下文无关文法、正则文法。由文法所生成的语言也相应分成这 4 类。自动机是能由文法 G 产生出 G 所生成的语言 L(G)的一种装置，它可用一个 5 元组表示。自动机也可分好多种，其中有限状态自动机是能由正则文法产生正则语言的自动机，即它对应于 3 型语言；下推自动机则对应于上下文无关语言(2 型)；线性界限自动机则对应于上下文有关语言(1)型；图灵机则对应于 0 型语言。

文法 G 是任意给出的，有可能出现这样的情况：给定某个文法 G，对 VT 中的终止符所组成的任何字符串都无法识别出它能由 G 生成，即 L(G)是个空集。也可能有某个给定的文法 G，其 L(G)是无限制。由于 VT 的终止符可重复出现，字符串长度无限制，因此不可能用列举终止字符串进行句法分析，以判断给定的 G，其 L(G)是无限集还是有限集，或是空集。

我们期望能有一种算法，可直接从 G 出发，通过有限步运算给出 L(G)是空集、有限集或无限集的结论。若存在这样的算法，就是可判定的，否则就不是可判定的。研究中发现可否判定 L(G)是空集、有限集或无限集与文法 G 有关的，可以证明，当文法 G 是 2 型或 3 型时，是可判定的。类似的，对二个文法 G1 与 G2 是否等价即是否有 L(G1)=L(G2)，只有当 G1 和 G2 都是 3 型文法时，才是可判定的。

【答案】A：④ B：② C：⑤ D：⑤ E：③

试题 9 (1994 年试题 2)

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

在下列程序中：

```
Program test (input, output);

Var i, j: integer;
```

```
Procedure calc (p1, p2: integer);

Begin p2:=p2*p2; p1:=p1-p2:=p2-p1;end{calc}

Begin{main}i:=2,j:=3

Calc(i,j);write(j);

End{main}
```

当参数传递采用引用方式(Call by reference)时，所得结果 j=A；

当参数传递采用换名方式(Call by name)时，所得结果 j=B；

当参数传递采用赋值方式(Call by value)时，所得结果 j=C；

递归是程序设计中很重要的一种控制结构，通常实现递归时，采用的数据结构是 D。

对那些既可以用递归方式，也可以用循环方式求解的问题，就执行效率而言 E。

供选择的答案

A～C：①0 ②3 ③5 ④6
⑤10 ⑥16 ⑦20 ⑧28

D：①数组 ②栈 ③队列 ④循环链表

E：①数组 ②两者相同 ③循环优于递归 ④递归优于循环

【解析】

一个过程的过程体若包含对其自身的调用，则称此过程是直接递归的。若一个过程的过程体调用某过程，而该过程又调用原过程或经一系列调用后又回到对原过程的调用，则称此原过程是间接递归的。通常实现递归时采用的数据结构是栈，这是因为栈有先进后出的特性，可以保存调用时的"现场"，并在调用结束时恢复"现场"，栈是实现递归的简单途径。对于既可用递归方式求解，也可用循环方式求解的问题，就执行效率和资源而言，显然是循环优于递归，因为递归的开销大。

当用户在调用点调用一个过程时，会通过参数传送信息，一个过程的形式参数用来向过程传送信息的标识符，实在参数用来在调用点向被调用过程传送信息。形式参数和实在参数之间的关系通常按位置来标定，不同程序语言所规定的参数信息传送方式不同。

当采用引用方式(Call by reference)或换名方式(Call by name)时，在过程中对形式参数的调用本质上是对实在参数单元的引用。先是给形式参数赋初值，而后，在过程中对该形式参数的赋值最终引起调用程序中实在参数值的改变。在本题中形式参数为 p1 和 p2。实在参数初值为 i=2 和 j=3，通过引用方式调用这两个参数，将执行以下计算过程：

p1=2,p2=3　　p2:=p2*p2=9　　p1:=p1-p2=2-3=-7　　p2:=p2-p1=9-(-7)=16

所得结果为 j=16。

参数传送采用赋值方式时，从调用点向被调用过程传送的是实在参数的值。这一值成为过程中相应位置上形式参数的初值，此后该形式参数在过程中实际是局部变量，其结果无需返回给实在参数。本题中实在参数 j=3，在过程中仅起向形式参数 p2 赋初值的作用。过程中关于 p2 的运算对 j 不再起作用，因而过程调用结束后 j 的值仍为 3。

【答案】A：⑥ B：⑥ C： ②D： ②E：③

试题 10 (1994 年试题 16)

从供选择的答案中，选出应填入下面有关形式语言叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

文法 G=(VT,VN,P,S)的类型由 G 中的 A 决定。若 G0=({a,b},{S,X,Y},P,S)，P 中的产生式及其序号如下：

1：S→XaaY

2：X→YY|b

3：Y→XbX|a

则 G0 为型文法，对应于 C，由 G0 推导出句子 aaaa 和 baabbb 时，所用产生式序号组成的序列分别为 D 和 E。

供选择的答案：

A：①：VT ②VN ③P ④S

B：①0 ②1 ③2 ④3

C：①图灵机 ②下推自动机 ③有限状态自动机 ④其他自动机

D、E：①13133 ②12312 ③12322 ④12333

【解析】

形式语言的文法是一个 4 元组 G=(VT，VN，P，S)，其中 VT 是非空有限集，称为终端符集，VN 也是非空有限集，称为变量集；P 为产生式集；S 为起始符，S∈Vn。形式语言的文法按 P 的特性可分成正则文法、上下文无关文法、上下文有关文法和图灵文法 4 种类型，又分别称为 3 型、2 型、1 型和 0 型文法。所对应的可实现的自动机分别为有限状态自动机、下推自动机、线性有界自动机和图灵机。

本题中给出的文法、生产式左部均是单个变量，因此是上下文无文法。由此文法导出句子 aaaaa 的产生式号的序列及推导过程如下：

S → XaaY 1

→ YYaaY 2

→ aYaaY 3

→ aaaaY 3

→ aaaaa 3

句子 baabbb 的推导过程为：

S → XaaY 1

→ baaY 2

→ baaXbx 3

→ baabbx 2

→ baabbb 2

因而产生式号的序列是 12333 和 12322。

【答案】：A：③ B：③ C：② D：④ E：③

试题 11 (1993 年试题 7)

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

根据乔姆斯基 50 年代建立的形式语言的理论体系，语言的文法被分成 4 种类型，即 0 型(短语文法)、Ⅰ(上下文有关文法)、Ⅱ型(上下文无关文法)和Ⅲ型(正规文法)。其中Ⅱ型文法与 A 等价，所以有足够的能力描述多数现今程序设计语言的语法缃结构。一个非确定的有限自动机必存在一个与之等价的 B。从文法描述语言的能力来说 C 最强，D 最弱。由 4 类文法的定义可知 E 必是Ⅱ型文法。

供选择的答案

A、B：①确定的有限自动机 ②图灵机
③非确定的下推自动机 ④非确定的有限自动机
⑤有限自动机 ⑥线性有界自动机

C～E：①0 型文法 ②Ⅰ型文法 ③Ⅱ型文法 ④Ⅲ型文法

【解析】

Ⅱ型文法与非确定的下推自动机等价，所以有足够的能力描述现在大多数程序设计语言的语法结构。一个非确定的有限自动机必存在一个与之等价的有限自动机。从文法描述语言的能力来说，0 型文法最强，Ⅲ型文法最弱。

语言的文法是一个 4 元组(Vt，Vn，P，S)，其中 Vt 是终结符号的非空有限集，Vn 是非终结符号的非空有限集，S 是一个特殊的非终对符号，P 是产生式的有限集，由Ⅲ型文法的定义：一个文法 G 是Ⅲ型文法，如果 G 是Ⅱ型文法，并且 G 的每个产生式为 A→σB 或 A→σ，其中 A∈Vt*，A，B∈Vn，可知，Ⅲ型文法必是Ⅱ型文法。

【答案】A：③ B：① C：① D：④ E：④

试题 12 (1992 年试题 2)

从供选择的答案中选出应填入{ }的正确答案，把编号写在答卷的对应栏内。

最早体现结构化程序设计思想的程序设计语言是 A，最早使用 BNF 文法定义程序设计语言语法的语言是 B。最早提出类(即 CLASS)的概念语言是 C，了早完备地体现面向对象并提出继承概念的程序设计语言是 D，最早的人工智能语言是 E。

供选择的答案：

A～E：①Ada ②Pascal ③Algol 68 ④Algol 60 ⑤Simula ⑥Lisp ⑦Prolog ⑧Smalltalk 80 ⑨C ⑩C++

【解析】

在 50 年代末和 60 年代初，由国际委员会设计了 Algol 语言，P.Naur 在随后的修改报告中用严格的 BNF(Backus-Naur Form)形式化方法定义了 Algol 语言的文法。

LISP 语言是由麻省理工学院的 John McCarthy 和他的研究小组最初约在 1960 年设计实现的。结构化程序设计思想是由 EW.Dijkstra 和 C.A.R.Hoare 等在近 60 年代末提出的。N.Wirth 提出的 Pascal 语言是第一个较系统地体现结构化程序设计思想的程序设计语言。

最早提出类(CLASS)概念的语言是 Simula；最早完备地体现面向对象并提出继承概念的程序设计语言是 Smalltalk 80；Lisp 是一种计算机的表处理语言，是最早也是应用最广泛的人工
智能语言。

【答案】A：② B：④ C：⑤ D：⑧ E：⑥

试题 13 (1991 年试题 7)

从供选择的答案中选出应填入列下叙述中 内的正确答案，把编号写在答卷的对应栏内。编译程序中语法分析器接受以 A 为单位的输入，并产生有关信息供以后各阶段使用。B、LR 分析法和 C 是几种常见的语法分析技术，其中 B 和 LR 分析法属于自下而上分析法，而 C 属于自上而下分析法。LR 分析法主要有 SLR(1)、LR(0)、LR(1)和 LALR(1)4 种，其中 D 的分析能力最强，E 的分析能力最弱。

供选择的答案

A：①：表达式 ②单词 ③产生式 ④语句

B、C：①递归下降法 ②算符优先法 ③语法制导翻译法
④数据流分类法 ⑤自动机分析法

D、E：①SLR(1) ②LR(0) ③LR(1) ④LALR(1)

【解析】

编译程序中语法分析器接受以单元为单位的输入，并产生有关信息供以后各阶段使用。算符优先法、LR 分析法和递归下降法是几种常见的语法分析技术，其中算符优先法和 LR 分析法属于自下而上分析法，而递归下降法属于自下而下分析法。LR 分析法主要有 SLR(1)、LR(0)、LR(1)和 LALR(1)等 4 种，其中 LR(1)的分析能力最强，LR(0)的分析能力最弱。

【答案】A：② B：② C：① D：③ E：②

2.3 操作系统基础知识

2.3.1 主要知识点

掌握操作系统的类型、功能、层次结构和进程概念，以及作业、处理机、存储、文件和设备等管理的原理和方法。

2.3.1.1 操作系统类型和功能

根据使用环境和对用户作业的处理方式划分，操作系统的基本类型可以分为批处理操作系统、分析操作系统和实时操作系统 3 大类型。

分时操作系统使多个用户同时以会话方式控制自己程序的运行，每个用户都感到似乎各自有一台独立的、支持自己请求服务的系统。

实时系统往往是专用的，系统与应用很难分离，常常紧密结合在一起。实时系统并不强调资源利用率，而更关心及时性(时间紧迫性)、可靠性和完整性。实时系统又分成实时过程控制与实时信息处理两种。

网络环境下的操作系统又分成网络操作系统和分布式操作系统。分布式操作系统要求一个统一的操作系统，负责全系统的资源分配和调度，为用户提供统一的界面。它是一个逻辑上紧密耦合的系统。而网络操作系统用户则需指明欲使用哪一台计算机上的哪个资源。

操作系统主要有 5 个功能模块：处理器管理、存储管理、设备管理、文件管理和用户接口。

2.3.1.2 进程和进程管理

(1)进程

进程是一个程序关于某个数据集的一次运行。也就是说，进程是运行中的程序，是程序的一次运行活动。相对于程序，进程是一个的概念，而程序是静态的概念，是指令的集合，因而进程具有动态性和并发性。

在操作系统中进程是进行系统资源分配、调度和管理的最小单位，注意，现代操作系统中还引入了线程(Thread)这一概念，它是处理器分配资源的最小单位。

(2)进程的状态及其转换

多道系统中，进程的运行是时走时停的。它在处理器上的交替运行，使它的运行状态不断地变化着，最基本的状态有 3 种，即运行、就绪和阻塞。

- 运行：正占用处理器。
- 就绪：只要获得处理器即可运行。
- 阻塞：正等待某个事件的发生。

(3)进程控制块

进程是一个动态的概念，在操作系统中，引入数据结构--进程控制块(简记为 PCB)来标记进程。PCB 是进程存在的唯一标志，PCB 描述了进程的基本情况。从静态的观点看，进程由程序、数据和进程控制块组成；从动态的观点看，进程是计算机状态的一个有序集合。

程序是进程运行所对应的运行代码，一个进程对应于一个程序，一个程序可以同时对应于多个进程，这个程序代码在运行过程中不会被改变，常称为纯码程序或可重入程序，他们是可共享的程序。

进程控制块保存进程状态、进程性质(如优先程度)、与进程有关的控制信息(如参数、信号量和消息等)、相应队列和现场保护区域等。进程控制块随着进程的建立而产生，随着进程的完成而撤消。

PCB 是操作系统核心中最主要的数据结构之一，它既是进程存在的标志和调度的依据，又是进程可以被打断并能恢复运行的基础。操作系统核心通过 PCB 管理进程，一般 PCB 是常驻内存的，尤其是调度信息必须常驻内存。

(4)进程管理

在操作系统中有许多进程，它们对应着不同的或相同的程序，竞争地使用着系统的资源。进程管理涉及到进程控制、队列管理和进程调度等。

进程的生命过程从它被创建时开始，直至任务终止而撤消，其间会经历各种状态的转换,它们都是在操作系统控制下完成的。操作系统提供了对进程的基本操作，也称为原语。这些原语包括创建原语、阻塞原语、终止原语、优先级原语和调度原语。

进程调度即处理器调度，它的主要功能是确定在什么时候分派处理器，并确定分给哪一个进程。在分时系统中，一般有一个确定的时间单位(时间片)。当一进程用完一个时间单位时，就发生进程调度，即让正在运行的进程改变状态并转入就绪队列的队尾，再由调度原语将绪队列的首进程取出，投入运行。

进程调度的方法基本上分为两类：非剥夺调度与剥夺调度。所谓非剥夺调度是指一旦某个作业或进程占有了处理器，别的进程就不能把处理器从这个进程手中夺走；相反，如果别的进程可将处理器从这个进程手中夺走则是剥夺调度。

进程调度的算法采用服务于系统目标的策略，对于不同的系统与系统目标，常采用不同的调度算法，如先来先服务、优先数调度和轮转法等。

(5)管程

管程是一种并发性的构造,包括用于分配一个特定的共享资源或一组共享资源的数据和过程。为了完成分配资源的功能，进程必须调用特定的管程入口。许多进程可能打算在不同的时间进入管程，但在管程边界上严格地实施互斥，在某一时刻，只允许一个进程进入。当管程中已有一个进程时，其他希望进入管程的进程必须等待。这种等待是由管程自动管理的。

管程中的数据或者是管程中所有的全局变量，或者是某个特定过程的局部变量。所有这些数据只能在管程内访问，在管程外的进程无法访问管程内的数据，这叫做信息掩蔽。

2.3.1.3 存储管理

现代计算机系统中的存储系统常是多级存储体系，至少有主存(内存)和辅存(外存)两级，有的系统有更多级数。主存大小由系统硬件决定，是实实在在的存储，它的存储容量受到实际存储单元的限制。虚拟存储(简称虚存)不考虑实际主存的大小和数据存取的实际地址，只考虑相互有关的数据这间的相对位置，其容量由计算机的地址的位数决定。

2.3.1.4 设备管理

设备管理是对计算机输入/输出系统的管理。其主要任务有：实现对外部设备的分配和回收；启动外部设备；控制输入/输出设备与处理器或主存间交换数据；实现对磁盘的调度；处理设备的中断；实现虚拟设备等。

外部和主存之间常用的传输控制方式有 4 种：程序控制方式、中断方式、直接存储访问(DMA)方式和通道方式。

2.3.1.5 文件管理

(1)文件系统

操作系统的文件系统包括两个方面：一方面包括负责管理文件的一组系统软件，另一方面包括被管理的对象文件。文件系统的主要目标是提高存储器的利用率，接受用户的委托，实施对文件的操作。主要问题是管理辅助存储器，实现文件从名字空间到辅存地址空间的转换，决定文件信息的存放位置、存放形式和存放权限，实现文件和目录的操作，提供文件共享能力和安全设施，提供友好的用户接口。

(2)文件的结构和组织
文件的结构是指文件的组织形式。从用户观点所看到的文件组织形式，称为文件的逻辑结构；从实现观点考查文件在辅助存储器上的存放方式，常称为文件的物理结构。

文件的逻辑组织是为了方便用户使用。一般文件的逻辑结构可以分为两种：无结构的字符流
文件和有结构的记录文件，后者也称为有格式文件。优化文件的物理结构是为了提高存储器的利用效率和降低存取时间。文件的存储设备通常被划分为大小相同的物理块，物理块是分配和传输信息的基本单位。文件的物理结构是指文件在存储设备上的存储主法。文件的物理结构涉及文件存储设备的组块策略和文件分配策略，决定文件信息在存储设备上的存储位置。

2.3.1.6 作业管理和用户界面

作业(Job)是系统为完成一个用户的计算任务或一次事务处理所做的工作的总和。操作系统中用来控制作业的进入、执行和撤消的一组程序称为作业管理程序，这些控制功能也能通过把作业步细化、通过进程的执行来实现。

用户的作业可以通过直接的方式，由用户自己按照作业步顺序操作；也可以通过间接的方式，由用户事先编写作业步依次执行的说明，一次交给操作系统，由系统按照说明依次处理。前者称为联机方式，后者称为脱机方式。

一般操作系统提供两种作业控制方式，一种为联机作业方式，另一种为脱机作业方式。联机作业方式是通过直接输入作业控制命令来提交和运行用户作业。脱机作业方式是通过作业控制语言(JCL，也称为作业控制命令)编写用户作业说明书。在这种方式中，用户不直接干预作业的运行，而是把作业与作业说明书一起交给系统(称为提交)。

作业调度主要是从后备状态的作业中挑选一个(或一些)作业投入运行。根据不同的调度目标，有不同的算法。作业调度算法有许多种，它们与进程调度相似，有的适宜于单道系统，有的适宜于多道系统。它们是先来先服务(FCFS)、短作业优先(SJF)、响应比(HRN)高者优先和成先级调度等。

2.3.1.7 其他管理

(1)死锁问题

如果一个进程正在等待一个不可能发生的事件，则称该进程处于死锁状态。系统发生死锁是指一个或多个进程处于死锁状态。产生死锁的主要原因是共享的系统资源不足，资源分配策略和进程的推进顺序不当。系统资源既可能是可重用的永久性资源，也可能是消耗性的临时资源。处于死锁状态的进程不能继续运行又占有了系统资源，阻碍其他进程的运行。对待死锁的策略

主要有：

①死锁的预防。不让任一产生死锁的必要条件发生就可以预防死锁。

②死锁的避免。这种策略不对用户进程的推进顺序加以限制，在进程申请资源时先判断这次分配安全否，只有安全实施分配，典型的算法是银行家算法。

③死锁的检测。这种策略采用资源请求分配图的化简方法来判断是否发生了不安全状态。资源请求分配图是一种有向图，表示进程与资源之间的关系。死锁的检测是在需要的时刻执行的，当发现系统处于不安全状态时，即执行死锁的解除策略。

④死锁的解除。解除死锁的基本方法是剥夺。一种方法是把资源从一些进程处剥夺分给别的进程，被剥夺资源的进程则需回退到请求资源处重新等待执行；另一种主法是终止一个进程，剥夺其全部资源，以后再重新运行被终止的进程。

(2)多重处理器系统与线程

多重处理系统的主要目标是为了提高系统的处理能力，也是为了提高系统的可靠性。多重处理系统的操作系统除了具有单处理器操作系统的功能以外，还应提供处理器的负载平衡、处理器发生故障后的结构重组等功能。一般多重处理系统的操作系统可以分为主从式、分离执行式和移动执行式 3 类。

对称多处理器系统 SMP 是由若干同构甚至相同的处理器构成的一个系统。Solaris 和 Windows NT 等操作系统支持 SMP 系统。操作系统提供了线程(Thread)机制以发挥多个处理器的作用。在多线程系统中，一个进程可以由一个或多个线程构成。进程是资源分配的基本单位，也是被保护的基本单位。一个进程对应于一个保存进程映象的虚地址空间，每一线程可以独立运行一个进程的线程共享这个进程的地址空间。有多种方法可以实现多线程系统，一种方法是核心级线程，另一种方法是用户级线程，也可以把两者组合起来。

2.3.1.8 操作系统的结构

(1)无序结构法，又称整体结构或模块组合结构。它以大型表格和队列为中心，操作系统的各部分程序围绕着表格运行，整个系统是一个程序。这种操作系统常称为面向过程的操作系统。

(2)层次结构法是把一个大型复杂的操作系统分解成若干个单向依赖的层次，由多层的正确性保证操作系统的可靠性。层次结构清晰，且有利于系统功能的增加或删改。

(3)面向对象的操作系统基于面向对象程序设计的概念，采用了各种不同的对象技术。在计算机系统中对象是操作系统管理的信息和资源的抽象，是一种抽象的数据类型。可以把对象作为系统中的最小单位，由对象、对象操作、对象保护组成的操作系统，就是面向对象的操作系统。如 Windows NT 中有执行体对象(进程、线程、文件和令牌等)和内核对象(时钟、事件和信号等)。

(4)微内核结构法把系统的公共部分抽象出来，形成一个底层核心，提供最基本的服务，其他功能以服务器形式建立在微内核之上。它具有良好的模块化和结构化特征，模块之间和上
下层之间通过消息来通信。建立在微内核上的服务器可以根据不同的需要构造，从而形成不同的操作系统，如 Windows NT 操作系统。

2.3.1.9 常用操作系统

UNIX 系统是一个分时操作系统。它利用最内层硬件提供的基本服务，向外层提供全部应用程序所需要的服务。应用程序组可以构成应用子系统，如 UNIX 系统的源代码控制系统(SCCS)、图形(X-Window、Motif)等。

Windows NT 系统是 20 世纪 90 年代的操作系统技术，适用于高档工作站平台、局域网服务器或者主干计算机。Windows NT 支持对称处理器结构，支持多线程

并行，采用 90 年代操作系统技术(即微内核技术)，在体系结构上采用客户机/服务器模式。

2.3.2 试题解析

操作系统也是每年必考的知识点。从历年试题统计(见表 2-4)来看，考查内容主要集中在存储管理、进程、作业管理等知识点，特别是有关进程的内容反复考查，考查的问题也都差不多。操作系统基础试题还有一个突出的特点，自 1998 年以来,每年都有一道试题与过去考查过的试题基本相同,其中 1998 年试题 7 与 1990 年试题 2 基本相同，1999 年试题 4 与 1995 年试题 1 基本相同，2000 年试题 2 与 1998 年试题 9 几乎完全一样，1991 年试题 6 则原封不动地搬以了 2000 年程序员级上午试卷上。由此可见，复习历年试题是非常重要的。

表 2-4 历年高级程序员级操作系统基础知识试题统计

试题	考查知识点
1990 年试题 2	虚拟存储管理系统
1991 年试题 2	操作系统常识，涉及重定向和批处理（2000 年程序员级上午试题 4 重考）
1991 年试题 6	存储管理
1992 年试题 4	进程
1993 年试题 3	进程间通信，涉及信号量与 P、V 操作
1993 年试题 15	管程
1994 年试题 4	进程的概念和性质
1994 年试题 16	文法
1995 年试题 1	进程
1996 年试题 6	作业管理
1997 年试题 2	作业管理
1998 年试题 4	涉及进程间通信，主要考查信号量与 P、V 操作
1998 年试题 7	虚拟存储管理（基本同 1990 年试题 2）
1998 年试题 9	段页式存储管理
1999 年试题 4	进程（基本同 1995 年试题 1）
2000 年试题 2	段页式存储管理（同 1998 年试题 9）
2000 年试题 4	进程与线程

试题 1 (2000 年试题 2)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

当存储器采用段页式管理时，主存被划分为定长的 A，程序按逻辑模块分成 B。在某机器的多道程序环境下，每道程序还需要一个 C 作为有用户标志号，每道程序都有对应 D。一个逻辑地址包括 C，x、段号 s、页号 p 和页内地址 d 等 4 个部分。

设逻辑地址长度分配如下，其中 x、s、p、d 均以二进制数表示。

21 22	19 14	13 11	10 0
x	s	p	d

其转换后的地址为 E。

供选择的答案

A：①段 ②页 ③区域 ④块

B：①区域 ②页 ③块 ④段

C：①模块号 ②区域号 ③基号 ④区域

D：①一个段表和一个页表 ②一个段表和一组页表
③一组段表和一个页表 ④一组段表和一组页表

E：①$x*2^{20}+s*2^{14}+p*2^{11}+d$ ②(((x)+s)+p)+d
③(((x)+s)+p)*2^{11}+(d) ④(((x)+s)+p)*2^{11}+d

【解析】

本题涉及存储管理知识，主要考查段页式存储管理，与 1998 年试题 9 几乎完全相同。

段页式存储组织综合了段式组织与页式组织的特点，主存被划分为定长的页，段页式系统中的虚地址形式是(段号、页号、位移)。系统为每个进程建立一个段表，为每个段建立一个页表。也就是说，先将程序按逻辑模块(如主程序、子程序和数据段等)分为若干段，再将每个段分为若干页。

对于多道程序环境，每道程序有一个基号与其他程序相区分，每道程序可以有多个段，但只有一个段表，每个程序可以有多个页表。

段页式存储体系中逻辑地址与物理地址的转换：首先由基号段号得到段表的地址，再访问段表得到页表的地址，再由页表得到物理块的地址，此时得到的地址是高 11 位的地址，因此需乘以 2^{11} 再加上页内地址，才得到真正的物理地址。

【答案】A：② B：④ C：③ D：② E：④

试题 2 (2000 年试题 4)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

A 是操作系统中可以并行工作的基本单位，也是核心调度及资源分配的最小单位，它由 B 组成，它与程序的重要区别之一是 C。

在 SMP 系统中，操作系统还提供了 D 机制，它是 E 的最小单位。
供选择的答案

A：①作业 ②过程 ③函数 ④进程

B：①程序、数据和标识符 ②程序、和 PCBD；③程序、标识符和 PCB ④数据、标识符和 PCB

C：①程序可占有资源，而它不可 ②程序有状态，而它没有
③它有状态，而程序没有 ④它能占有资源，而程序不能

D：约束 ②线程 共享 分时

E：①存储器分配 ②资源分配 ③处理器分配 ④网络结点分配

【解析】

本题考查关于进程与线程方面的知识。

进程是一个程序关于某个数据集的一次运行。也就是说，进程是运行中的程序，是程序的一次运行活动。相对于程序，进程是一个动态的概念，而程序是静态的概念，是指令的集合，因而进程具有动态性和并发性。

进程是一个动态的概念，在操作系统中，引入数据结构-PCB(进程控制块)来标记进程，。PCB 是进程存在的唯一标志，PCB 描述了进程的基本情况。从静态的观点看，进程由程序、数据和进程控制块组成。

对称多处理器系统 SMP 是由若干同构甚至相同的处理器构成一个系统，其操作系统提供了线程(Thread)机制以发挥多个处理器的作用在多线程系统中，一个进程可以由一个或多个线程构成。进程是资源分配的基本单位，而线程是处理器分配的最小单位。

【答案】：A：④ B：② C：③ D：② E：③

试题 3 (1999 年试题 4)

从供选择的答案中，选出应填入下面叙述中{ }内的最解切的解答，把相应编号写在答卷的对应栏内。

进程是操作系统中的一个重要概念。进程是一个具有一定独立功能的程序在某个数据集合上的一次 A。

进程是一个 B 的概念，而程序是一个 C 的概念。

进程的最基本状态有 D。在一个单处理机中，若有 6 个用户进程，在非管态的某一时刻，处于就绪状态的用户进程最多有 E 个。

供选择的答案

A：①单独操作 ②关联操作 ③运行活动 ④并发活动

B：①静态 ②动态 ③逻辑 ④物理

C：①物理 ②逻辑 ③动态 ④静态

D：①就绪、运行、隐蔽 ②停止、就绪、运行
③运行、就绪、阻塞 ④就绪、撤消、运行

E：①5 ②6 ③1 ④4

【解析】

本题考查操作系统中的进程知识，与 1995 年试题 1 基本相同。

进程是操作系统中基本的并行单位、资源分配单位和调度单位。通常，进程可分为用户进程和系统进程两类，前者控制用户作业的运行，后者完成系统内部分工的管理工作。

从静态的角度看，进程由程序、数据和进程控制块(JCB)组成；从动态的角度看，进程是计算机状态的一个有序集合。进程是一个具有一定独立功能的程序在某个数据集合上的一次运行，其中可能要涉及多个程序；而一个程序的运行过程中可能有若干进程依次或并行活动。

进程既是一个运行单位，又是一个调度单位和资源使用单位。进程和程序是有区别的。进程是有状态的，而程序没有状态。进程是一个动态的概念，它可以执行、暂停、继续运行，而程序是一个静态的概念，它体现了某个算法，而且多个进程可以共享一个程序。

进程的基本状态有就绪、运行和阻塞 3 种。阻塞态是指一个进程由于某种原因不具备运行条件时所处的状态，这时它必须等待，引起等待的条件一旦清失，进程便具备了运行的条件，状态转变为就绪态；就绪态是指一个进程具备了运行的条件，但由于没有占有处理机而不能运行所处的状态，一旦处于就绪态的进程轮到该进程占有处理的时间片或处理机空闲，其状态就转变为运行态，投入运行；运行态是指一个进程正占用着处理机时的状态，这时，处理机正在执行该进程的程序，运行过程中进程会因时间片已到等非资源请求原因退出运行转变为就绪态，因资源请求原因而不具备运行条件时，该进程的状态就要转变为阻塞态。

由于只有占有处理机进程才能处于运行态，所以在一个单处理机中，非管态的某一时刻，系统中处于就绪态或阻塞的进程可能有多个，但处于运行态的进程最多只有一个，也有可能是 0 个（系统死锁），这样处于就绪态的进程数只能是进程总数减 1，不可能出现全部处于就绪态而无运行态进程的情况。

【答案】A：③ B：② C：④ D：③ E：①

试题 4 (1998 年试题 5)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

信号量是操作系统中用作互斥和同步机制的一个共享的整数变量。信号量仅可以由初始化、唤醒（Signal）和等待（Wait）3 种操作访问。

对于给定的信号量 S，等待操作 Wait(S)（又称 P 操作）定义为 IF S 大于 0 THEN A ELSE 挂起调用的进程:唤醒操作 Signal(S)(又称 V 操作)定义为 IF 为\存在等待的进程 THEN 唤醒这个进程 ELSE B 。
给定信号量 S，要以定义一个临界区来确保其互斥，即保证在同一时刻这个临界区只能够被一个进程执行。当 S 被初始化为 1 时，代码段定义了一个临界区。

C:
（临界区）
D

这样的临界区实际上是将共享数据和对这些数据的操作一起封装起来，通过其互斥机制，一次只允许一个进程进入，这种临界区能常称为 E 。

供选择的答案

【解析】

信号量是操作系统中用作互斥和同步机制的一个共享整数变量，除初始化外，仅能通过两个标准的原子操作（Atomic Operation）-Wait(s)和 Signal(S) 来访问。对于给定的信号量 S，等待操作为：若 S＞则 S：=S-1，禁止其他进程访问此临界资源，否则挂起调用的进程。唤醒操作为：若存在等待的进程，则唤醒它，否则 S：=S+1，允许其他进程访问此临界资源。

每个进程中访问昨界资源的那段代码称为临界区（Critical Section ）。显然，若能保证每个进程互斥地进入自己的临界区，就能实现它们对临界资源的互斥访问。这样，每个进程在进入临界区，。访问该资源，并设置信号量，表示资源正在被访问，否则应等待（挂起）， 这个操作即 Wait(S)。当其访问完临界资源，退出临界区时，检查若有进程被挂起（即在等待访问比临界资源），则唤醒该进程，否则应当恢复信号量，以使其他地程将来能访问此临界资源，这个操作即 Signal(S)。

引入信号量机制的目的是为了消除与时间有关的错误，但如果在使用 Wait(S)和 Sigal（S）操作时出现错误，同样也会出现与时间有关的错误。为了解决这种问题，Dijkstra 在 1971 年提出，把所有进程对某一种临界资源的同步操作都集中起来，构成一个所谓的"秘书"进程，凡是要访问此临界资源的进程都要先报告"秘书"，由"秘书"来实现者进程的同步。后来，"秘书"进程思想又进一步发展为管程的概念，即"一人管理定义了一个数据结构和能为并发进程所执行（在此数据结构上）的一组操作，这组操作能同步改变管程中的数据"。

【答案】A：③ B：② C：⑧ D：⑦ E：③

试题 5 （1998 年试题 7）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

虚拟存储管理系统的基础是程序的 A 理论。这个理论的基本含义是提程序执行时往往会 B 访问内存储。程序的 A 表现在 C 和 D 上。C 是指最近被访问的存储单元可能马上又要被访问。 D 是指马上被访问的单元，而其附近的单元也可能马上被访问。

根据这个理论，Denning 提出了工作集理论。工作集是进程运行时被频繁也访问的页面集合。在进程运行时，如果它的工作页面都在 E 器内，能够使用进程有效地运行，否则会出现频繁的页面调入/调出现象。

供选择的答案

A： ①局部性 ②全局性 ③动态性 ④虚拟性

B： ①频繁地 ②均匀地 ③不均匀地 ④全面地

C、D： ①数据局部性 ②空间局部性 ③时间局部性 ④数据全局性

E： ①外部存储 ② 主存储 ③辅助存储 ④虚拟存储

【解析】

本题考查虚拟存储管理系统知识，与 1990 年试题 2 基本相同。

虚拟存储管理系统的基础是程序的局部性原理。所谓程序局部性原理是指程序在执行时所呈现的局部性规律，即在一段较短时间内，程序的执行仅限于某个部分。相应地，它所访问的存储器空间也局限在某个空间。局部性原理又是表现为两个方面：

（1）时间局限性

如果某条指令被执行，则不久以后该指令很可能再次被执行；如果某条数据结构被访问，则不久以后该数据结构很可能再次被访问。产生时间局限性的主要原因是程序中有大量的循环操作。

（2）空间局限性

一旦程序访问了某个内存单元，不久以后，其附近的内存单元也要被访问，即程序在一段时间内所访问的存储器空间可能集中在一定的范围之内，其最常见情况就是程序的顺序执行。

工作集是指在某段时间隔内，进程所要访问的页面集合。虽然程序只需少量的几页内存就可以运行，但为了使程序更有效地运行，必须使程序的工作集全部在内存（主存储器）当中，否则会使进程在运行中频繁出现缺页中断，从而出现频繁的页面调入/调出现象。

【答案】A：① B： C：② D：③ E：④

试题 6 （1998 年试题 9）

从供选择的答案中，选出填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

在段页式管理的存储器中，实存等分为 A ，程序按逻辑模块分成 B 。在多道程序环境下，每道程序还需要一个 C 作为用户标志号。每道程序都有对应的 D 。一个逻辑地址包括 C X、段号 S 页号 P 和页内地址 d 这 4 个部分。

假设总长度为 22 位的逻辑地址格式分配为：21～20 位 X，19～14 位 s,13～位 P,10～0 位 d。若 x、s、p 和 d 均以二进制数表示，其转换成的物理地址为 E 。

供选择的答案

A～C ①段 ②页 ③基 ④模块
⑤区域 ⑥段号 ⑦页号 ⑧基号
⑨模块号 ⑩区域号

D： ①一个段表和一个页表 ②一个段表和一组页表
③一组段表和一个页表 ④一组段表和一组页表

E： ①$xx2^{20}+sx2^{14}+px2^{11}+d$ ②（（x）+s+p）$x2^{11}+d$
③((x)+s+p)+d ④((x)+s)+p)$x2^{11}+d$
（注：式中（Y）表示地址为 Y 的单元的内容。）

【解析】

请参见本节试题 1（2000 年试题 2）的解答。

【答案】A：①B：② C：③ D： E：④

试题 7 （1997 年试题 2）

从供选择的答案中，选出应填入下面叙述中{} 内的最确切的解答，把相应编号写在答卷的对应栏内。

在有一台处理机 CPU 和两台输入车输出设备 IOI 和 IO2，且能够实现抢先式多任务并行工作的多道程序环境内，投入运行优先级由高到低的 P1、P2 和 P3P 这 3 个作业。它们使用设备的先后顺序和

占用设备时间分别是：

作业 P1：IO2（30ms）、CPU（10ms）、IOI(30ms)、CPU(10ms)

作业 P2：IOI (20ms)、CPU（20ms）、IO2(40ms)

作业 P3：CPU (30ms)、IOI (20ms)

假设对于其它辅助操作时间可以忽略不计，作业 P1、P2、P3 从投入到完成所用的时间分别是 A ms、 B ms、 C ms。3 个作业从投入运行到全部完成，CPU 的利用率约为 D %，IO1 的利用率约为 E %

假设在系统中仅有这 3 个作业投入运行，各设备的利用率是指该设备的使用时间同作业进程组全部完成所占用量长时间的比率。

【解析】

处理机和输入输出设备采取抢先式多任务并行工作方法，分析完成每个作业所需的时间时，可先分析优先高级的作业。高优先级的作业有优先获取资源的优势，在只有 3 个作业时，最高优先级的作业 P1 发出的申请设备请求会立即得到响应。各设备占用时间为：

在 P1 优先占用设备的基础上，P1、P2 和 P3 可以在剩下的作业中优先得到资源。

从表 2-5 可以看出 P2 在使用 IOI 设备 20ms 后，要使用 CPU20ms，但当其运行 10 ms 后，由于 P1 优先高级并采用抢先方式，所以将 P2 暂停，让 P1 先运行。同理，P3-开始就使用 CPU，但在运行 20ms 后，要让给 P2 和 P1。

P1 投入运行到完成需要 80ms，P2、P3 由行等待资源，运行时间都延长为 90ms。CPU 在 60ms～70ms 和 80ms～90ms 的共 20ms 时间内没有利用，所以利用率为 70/90≈ 78%，IO2 的利用率也为 78%。

表 2-5 1997 年试题 2

IO1	P_2	P_2			P_3	P_3	P_3	P_3	P_3
IO2	P_1	P_1	P_1			P_2	P_2	P_2	P_2
CPU	P_3	P_3	P_2	P_1	P_2	P_3		P_1	

试题 8 （1996 年试题 6）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

由于内存大小有限，为使得一个或多个作业能在系统中运行，常需要用外存来换取内存。其中作业为单位对外存进行交换的技术称为 A 技术，在作业内部对内外存进行交换的技术称为 B 技术。

用我存换内存是以牺牲程序运行时间为代价的。为提高 CPU 有效利用率，避免内外存的频繁交换， B 技术常用某种 C 来选择换出内存的页面，它的基础是程序的 D 。据此，操作系统可根据 E 来改善系统的性能。 是一个进程在定长的执行时间内涉及到的页面集合。

供选择的答案

A、B ①SPOOLING ②SWAPPNG ③虑拟存储 ④虑拟机
⑤进程管理 ⑥设备管理

C： ①页面分本策略 ②页面标志策略 ③页面淘汰策略 ④虑拟机

D： ①完整性 ②局部性 ③递归性 ④正确性

E： ①工作集 ②页面集 ③段号集 ④元素集

【解析】

SWAPPING 技术是对换术的一种，用于解决由于内存不足而无法运行多道程序的问题，它以整个进程（作业）为单位进行对换，被对换到外存的作业将在一段时间内停止运行，这种对换广泛应用于分时系统中。所以问题 A 选择②。

对于一道作业来说，受内存容量的限制，作业进程能得到的内存空间会小于作业所需的空间。为使作业能在系统中运行，需要采用作业内部的对换技术，在作业内部以“页”或“段”
为单位进行部分对换，即在内存中保留作业进程的一部分，而在外存放置作业进程的副本。
作业运行时可以随机地访问在内存的那部分程序和数据，一旦访问不在内存的程序或数据时，
就腾出部分暂时不用的内存区域，将它们的内容调出至外存，再将所要访问的那些内容调入
内存。实现作业内部的部分对换，从而允许作业的地址空间大于实际分配的内存区域，这就
是虚拟存储技术。所以问题 B 选择③。

虑拟存储技术中一个重要问题是如何选择页面置换算法，即如何淘汰页面，以避免内外存的频繁交换，降低系统性能，所以问题 C 选择③。

虑拟存储管理系统的基础是程序的局部性原理。所谓程序局部性原理是指程序在执行时所呈现的局部性规律，即在一段较短的时间内，程序的执行仅限于某个部分。相应地，它所访问的存储器空间也局限于某个空间。

程序局部性有两个方面的含义：时间局部性，即如果一条指令被执行，则在不久以后可能再次执行，产生时间局限的主要原因是程序中有大量的循环操作：空间局部性，即一旦程序访问了某个内存单元，不久以后，其附近的内存单元也要被

访问。程序在一段时间内所访问的存储器空间可能集中在一定的范围之内，其最常见的情况就是程序执行。所以问题 D 选择②。

操作系统可以利用工作集模型来改善系统的性能，工作集是指在某段时间间隔内，进程所要访问的页面集合。虽然程序只需少量的几页内存就可以运行，但为了使程序有效地运行，必须使程序的工作集全部在内存（主存储器）当中，否则会使进程在运行中频繁出现缺页中断，从而出现频繁的页面调入/调出现象。所以问题 E 选择 ①

【答案】A：② B：② C：③ D：④ E：①

试题 9（1995 年试题 1）

从供选择的答案中,选出应入下面叙述中{ }内的正确答案,把编号写在答卷的对应栏内.

在操作系统中,进程是一个具有一定独立功能的程序,在某个数据集合上的一次 A ，进程是一个 的 B 的概念，而程序是一个 C 的概念。

在一单处理机中，若有 5 个用户进程，在非管态的某一时刻，处于就绪状态的用户进程最多有 D 个， 最少有 E 个。

供选择的答案：

A：①并发活动 ②运行活动 ③单独操作 ④关联操作

B、C：①组合态 ②并联态 ③运行态 ④等待态
⑤静态 ⑥动态

D、E：①1 ②2 ③3 ④4

【解析】

本题考查进程知识，请参见本节的试题 3（1999 年试题 4）的解答。

在非管态的某一时刻，处理机将处理用户进程，显然应该而且必须有一个进程处于运行态，那么处于就绪态的进程最多只能有 4 个，最少有 0 人，即除了处于运行态的进程外，其余 4 个进程都不具备运行的条件而处于待态，使处于就绪态的进程只有 0 个。

试题 10 （1994 年试题 4）

从下面有关进程的概念和性质量叙述中，选出 5 条正确的叙述，并把编号按从小到大的次序写在答卷的 A～E 栏内。

①唤醒：挂起→就绪。

②封锁：就绪→挂起。

③调度：就绪→运行。

④超时：运行→挂起。

⑤超时：运行→就绪。

⑥用户进程可激发调度进程。

⑦用户进程可激发唤醒进程。

⑧用户进程可激发超时进程。

⑨进程接近 CPU 可接纳的限度时，可降低页面出错的频率。

⑩进程具有引用局部性时，可降低页面出错的频率。

【解析】

操作系统中的处理器管理主要是把 CPU 的处理时间合理地分配给进程，充分利用处理器的运算能力。进程就是系统进行分配和调度的最小单位，有等待、就绪和运行 3 种基本状态。等待状态时不具备运行条件，等待自身状态或系统状态变化；就绪状态时进程已分到资源。

进程的状态是可变化的，处于等待状态的进程只有在所等待的时间到来或条件满足时，才能进入就绪状态，这就是唤醒，因此①和⑦是正确的。就绪状态的进程被操作系统调度并选中便进入就绪状态，③是正确的，运行状态下，CPU 时间片用完，这就是超时，此时进程转入就绪状态， ⑤是正确的，④是错误的。运行状态只有在进程受阻的情况下才会进入等待状态，就绪状态不会因陋就简封锁进入挂起状态，②是错误的。从进程基本状态的转换来看，只有唤醒是受时间激发的，其他都是由操作系统进行内部管理，因此用户不能激发调度与超时，⑥和⑧是错误的。操作系统对存储的管理主要是对内存的分配、保护和扩充。内存分配管理办法之一就是分页式管理。当进程具有引用局性时可降低页面出错频率，可见⑩正确，⑨错误。

[答案]A：① B：③ C：⑤ D：⑦ E：⑩

试题 11 （1993 年试题 3）

从供选择的答案中，选出应填入下面关于操作系统叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

在操作系统中，解决进程间的 A 两种基本关系，往往运用对信号量进行 B 的 C 。例如，为保证系统数据库的完整性，可以把信号量定义为某个库文件（或记录）的锁，初值为 1，任何进程存取该库文件（或记录）之前先对它作一个 D ，

存取之后对它作一个 E，，从而做到对该文件（或记录）任一时刻只有一个进程可存取，但要注意使用不当引起的死锁。

供选择的答案

A： ① 同步与异步 ②串行与并行 ③调度与控制 ④同步与互斥

B： ① 消息操作 ②P-V 操作 ③开关操作 ④读写操作

C： ① 能信原语 ②调度算法 ③分配策略 ④进程控制

D、E ① 联机操作 ②操作 ③输出操作 ④读操作

⑤写操作 ⑥P 操作 ⑦输入操作

【解析】

在操作系统中，进程是可以独立运行的单位，由于进程同处于一个系统中，它们之间不可避免地会有某种联系，同步和互斥是进程间的两种基本关系。同步是指协调关系，即为了完成同一任务，两个进程应有某种约定，彼此协调运行速度：互斥是指竞争关系，即排它性。

当一个进程使用某一资源时，另一进程或者等待，或者根据某种优先级算法确定该资源归谁使用。 为了解决进程间的同步和互斥关系，可引用信号量的概念，信号量是一种特殊的变量，它的表现形式是一个整型普量附加一个队列，它只能被 P 操作和 V 操作使用。P 操作操作只能顺序进行，也就是必须在一个操结束以后才开始第二个操作，利用 P、V 操作可以容易地实现进程之间的互斥。

一般 P 操作与 V 操作的定义如下所述：

P 操作：（1）S-1→S;(2)如果 S＜0，则该进程进入等待状态；否则继续进行；

V 操作：（1）S+1→S；（2）如果 S≥0，则唤醒队列中的一个等待进程。

其实，P 操作就是给进程分配某个特实资源，而 V 操作则是进程在用完该资源后交还给系统。S 信号量为负数时也就表示等待使用资源的进程的个数。这样，。可以保证在任保一时刻对数据库文件只有一个进程可以存取。值得注意的是，同一信号量上的 P、V 操作往往是成对出现的。P、操作使用不当会引起死锁。

【解答】

A：④ B：② C：① D：⑥ E ②

试题 12 （1993 年试题 15）

从供选择的答案中，选出应填入下面关于操作系统叙述中 内的正确答案，把编号写在答卷的对应栏内。

操作系统中，管程（Monitor）由管程名、局部于管程的变量说明、使用共享资源并在数据结要上进行操作的若干过程以及对变量赋初值的语句等 4 个基本部分组成。每一个管程管理 A 。任何一个进程请求使用某临界资源时，对于其他相关进程而言 B 。其方法是通过调用特定的 C 才能进入管程，然后能过管程中 D 使用临界资源。在执行中发现共享临界资源被告或条件不成立时，调用管程的进程必须等待使用临界资源的另一进程来唤醒。为了表示不同的等待原因，设置了条件变量，条件变量是 E 。

供选择的答案

A： ①一个临界资源 ②一类临界资源
③若干个临界资源 ④所有的临界资源

B： ①可以先后进入管程 ②必须互斥地进入管程
③可以同时进入管程 ④可以按任意次序进入管程

C： ①管程入口 ②访管指令 ③系统调用命令

D： ①使用临界资源的进程 ②访问管程的进程
③使用临界资源的一个过程 ④调用管程的进程

E： ①与普通变量相同的变量
②与普通变量不同的变量，条件变量不能取任何，只是一个排队栈
③与普能变量不同的变量，只能取真、假值，不能取 0，1…，n 值

【解析】

在实现一个管程时必须考虑互斥、同步和条件变量这 3 个问题。管程中对每一个条件变量都予以说明，条件变量与普通变量不同，它不能取任何值，只能是一个排队栈。

当有几个进程调用某管程时，仅允许一个进程进入管程，其他调用者必须等待，也就是申请进程必须互斥地进入管程。方法是可通过调用特定的管程入口进入管程，再通过管程中的一个过程来调用临界资源。当某进程通过调用请求访问某临界资源而未能满足时，管程调用相应同步原语使该进程等待，并将它排在等待队列上。当使用临界资源的进程访问完该临界资源并释放之后，管程又调用相应的同步原语唤醒等待队列中的队首进程。

【答案】A：① B： ② C：① D ：③ E：②

试题 14 （1992 年试题 4）

从供选择的答案中造出应填入{ }的正确答案，把编号写在答卷的对应栏内。

从静态角度看，进程是由 A 、B 和 C 3 个部分组成。用户可通过 D 建立和撤消进程。通常，用户进程被建立后，E 。

供选择的答案

【解析】

A：①JCB ②DCB ③PCB ④PMT

B：①程序段 ②文件体 ③I/O ④子程序

C：①文件描述块 ②数据空间 ③系统调用 ④过程调用

D：①函数调用 ②宏指令 ③系统调用 ④过程调用

E：①便一直存在于系统中，直到被操作人员撤消
②随着作业运行正常或不正常结束而撤消
③随着时间片轮转而撤消与建立
④随着进程的阻塞或唤醒而撤消与建立

【解析】

进程是操作系统中可以并发运行的最基本单位。进程实体是由进程控制块(PCB)、程序段和数据空间组成。从进程的运行来看，进程可并发运行的程序在其数据集合上的运行过程。
操作系统一般向用户提供命令级和系统调用级的服务。包括设备控制、文件处理，建立进程和撤消进程以及其他的实用进程等。能常，用户进程被建立后，随着作业运行的正常或非正常结束而撤消。

【答案】A：③ B：① C：② D：③ E：②

试题 15 （1991 年试题 2）

从供选择的答案中选出填入下列叙述中的{ }内的正确答案，把编号写在答卷的对应栏内。

某些操作系统把一条命令的执行结果输出给下一条命令，作为它的输入，并加以处理，这种机制称为 A ，使命令执行的结果不在屏幕上显示，而引向另一个文件，这种机制称为 B ，使命令所需要的处理信息不从键盘接收，而调用一个正文文件。这种机制称为 C 。操作系纺不从键盘逐条接收命令并执行，而调用一个正文文件，执行其中保存的一系列命令，这种方式属于 D 方式，编写这样的文件应符合 E 的语法规则。

供选择的答案

A：①链接 ②管道（线） ③输入重新定向 ④输出重新定向

B：①清屏 ②屏蔽显示 ③输出重新定向 ④管道（线）

C：①输入重新定向 ②读保护 ③管道（线） ④批处理

D：①系统生成 ②初始装入 ③批处理 ④管道（线）

【解析】

一些操作系统（例如 UNIX）采用管道（线）的概念实现两个进程之间信息流的传递。管道是能够连接一个读进程和一个写进程，使二者进行通信的共享文件。通过把一个进程产生的信息流同管道传递给另一个进程，可以实现把一条命令的执行结果输出给下一条命令处理。某些操作系统。具有让命令的执行结果不显示在屏幕上而引向另一个文件的机制，称为输出重新定向；让命令所需的处理信息不从键盘接收，而取自另一个文件的机制，称为输入重新定向。

用户一般通过键盘键入命令，启用操作系统。有时为了提高使用效率，按命令语言的语法规定一系列命令保存在一个正文文件内，让操作系统调用这个批处理文件，成批地执行其中的命令，就是所谓的批处理方式。

【答案】A：② B：③ C： ① D：③ E：④

试题 16 （1991 试题 6）

从供选择的答案中选写出下列叙述关系最密切的存储管理方法，把编号写在答卷的对应栏内。

A.支持多道程序设计，算法简间，但存储器碎片多。

B.能消除碎片多和紧缩处理时间长的缺点，支持多道程序设计，但不支持虑拟存储。

C.克服了碎片多紧缩处理时间长的缺点，支持多道程序设计，但不支持虚拟存储。

D.支持虚拟存储，但不能以自然的方式提供存储器的共享和存取保护机制。

E.允许动态连接和装入，能消除碎片，支持虑拟存储。

供选择答案

A～E ①段页式 ②非请求分页式 ③请求分页式
④可重定位分区 ⑤固定分区 ⑥单一连续分配

【解析】
存储管理是操作系统的一个重要组成部分，包括决定存储分配的策略、分配和回收各存储区域、登记各存储区域的状态表等功能。对于虑拟拟存储管理还应包括页面的分配、管理和调度等功能。

存储分配方法有如下几种：

（1）固定分区存储管理。这是支持多道程序设计的最简单存储管理方法，它把主存划分成若干个固定的和大小不同的分区，每个分区能够装入一个作业，分区的大小是固定的，算法简单，但是容易生产较多的存储器碎片。

（2）可重定位分区存储管理。这是克服固定分区碎片问题的一种存储分配方法，它能够把相邻的空闲存储空间合并成一个完整的空区，还能够整理存储器内各个作业的存储位置，以在到消除存储碎片和紧缩存储空间的目的。紧缩工作需要花费大量的时间和系统资源。

（3）存储管理。这是支持多道程序设计的存储管理方法，它将存储空间和作业的地址空间分成若干个等分部分，并且进行编号和建立映射关系。分页存储管理分为请求分页式在轮制进程被执行两种，克服了分区存储管理中碎片多和紧缩处理时间长的缺点。非请求分页式在分页式支持虑拟存储，当进程需要用到某个页面时将该页面调入主存，把那些暂时无关的页面留在主存外。请求分页的实现方法比非请求分页的实现方法复杂，它是以页为单位进行存储保护和共享，而不能以自然方式提供存储器的共享和存储保护机制。

（4）连续分区把所有用户区都分配给唯一的用户作业，当作业被调度到时，进程全部进入内存，一旦完成，所有主存恢复空闲，因此，它不支持多道程序设计。

（5）段页式存储管理。这是分段式和分页式结合的存储管理方法。段可以定义为一组逻辑信息，例如子程序、数组或数据区。每个作业的地址空间由一些分段构成，每段有名字，且占有连续的地址空间，段与段之间在物理位置上可以连续。分段管理可以通过的移动来消除碎片，同时支持虑拟存储，实现动态连接和装配。分段法和分页法的结合目的在于可以使用分段法来分配和管理虑拟存储，而使用分页法来分配和管理主存储。把每个分段分成若干页面，借助物理方法管理这些页面，而不理管理一个分段，这样可充分利用分段管理和分页管理的优点。

【答案】A：⑤ B：④ C：② D：③ E：①

试题 17 （1990 年试题 2）
从供选择的答案中选出应填入下列叙述中的{ }内的正确答案，把编号写在答卷的对应栏内。

虑拟存储管理系统的基础是程序的局部性理论。此理论的基要含义是 A 。局部性有两种表现形式：时间局部性和 B 。它们意义分别为 C 和 D 。根据局部性理论，Denning 提出了 E。

供选择的答案
A、B：①程序执行时对主存的访问是不均匀的 ②代码的顺序执行
③变量的连续访问 ④指令局部性
⑤数据局部性 ⑥空间局部性

C、D： ①最新被访问的单元，很可能在不久的将来还要被访问。
②最新被访问的单元，很可能它附近的单元也即将访问。
③结构化程序设计，很少出现转移语句。
④程序中循环语句的执行时间一般很长。
⑤程序中使用的数据局部于各子程序。

E： ①Cache 结构的思想 ②工作集理论
③最近最少使用（LPU）页面置换算法
④先进先出（FIFO）页面置换算法

【解析】本题考查虚拟存储管理系统知识，请参见本节的 试题 5 的答案。

[答案]A：① B：⑥ C：① D：② E：②

2.4 软件工程基础知识

2.4.1 主要知识点

软件工程是计算机软件的一个重要分支，主要应掌握软件工程的基本原理以及软件设计与测试方法。

2.4.1.1 软件生存周期各阶段的任务
软件生存周期指出由软件定义、软件开发和软件维护等阶段组成的全过程。

（1）软件定义阶段

软件定义阶段主要解决软件要"做什么"的问题，也就是要确定软件的处理对象、软件与外界的接口、软件的功能、软件的性能、软件的界面以及有关的约束和限制。软件定义阶段通常可分成系统分析、软件项目计划和需求分析等阶段。系统分阶析的任务是确定待开发软件的总体要求和适用范围，以及与之有关的硬件和支撑软件的要求，该阶段所生产的文档可合并在软件项目计划阶段的文档（项目计划书）中。软件项目计划的任务是确定待开发软件的目标，对其进行可行性分析，并对资源分配、进度安排等做出合理的计划，该阶段所产生的文档有可行性分析报告和项目计划书。需求分析的任务是确定待开发软件的功能、性能、数据和界面等要求，从而确定系统的逻辑模型。该阶段产生的文档是需求规格说明书。

（2）软件开发阶段

软件开发阶段主要解决软件"怎么做"的问题，包括数据结构和软件结构的设计、算法设计、编写程序和测试，最后得到可交付使用的软件。软件开发阶段通常可分成软件设计、编码、软件测试等阶段。软件设计通常还可分成概要设计和详细设计。概要设计的任务是模块分解，确定软件结构、模块的功能和模块的接口，

以全国数据结构的设计。详细设计的任务是设计每个模块的实现细节和局部数据结构。设计阶段产生的文档有设计说明书，它也可分为概要设计说明书和详细设计说明书。编码的任务是用某种程序语言为每个模块编写程序，产生的文档有程序清单。软件测试的任务是发现软件中的错误，并加以纠正，产生的文档有软件测试计划和软件测试计划和软件测试报告。

（3）软件维护

软件维护任务就是为使软件适应外界环境的变化，进一步实现软件功能的扩充和质量的改善而修改软件。该阶段产生的文档有维护计划和维护报告。

2.4.1.2 软件开发模型

用不同的方式将软件生存周期中所有开发活动组织起来，形成不同的软件开发模型。常见的软件开发模型有瀑布模型、螺旋模型和喷泉模型等。瀑布模型给出了软件生存周期各阶段的固定顺序，上一阶段完成后才能进入到下一阶段。各阶段结束后，都要进行严格的评审。

2.4.1.3 结构化分析和设计方法

（1）结构化分析（SA）方法

结构化分析（SA）方法是一种面向数据流的需求分析方法，它适用于分析大型数据处理系统。结构化分析方法的基本思想是自顶向下逐层分解，把一个问题分解成若干个小问题，每个小问题再分解成若干个更小的问题，经过多次逐层分解，每个最低层的问题都是足够简单、容易解决的，这个过程就是分解的过程。SA 方法的分析结果由数据流图 DFD、数据词典和加工逻辑说明几个部分组成。

（2）结构化设计（SD）方法

结构化设计（SD）方法是一各面向数据流的软件设计方法，它可以与 SA 方法衔接，SD 方法采用结构图（SC）来描述程序的结构。结构图的基本成分由模块、调用和输入/输出数据组成。在需求分析阶段，用 SA 方法产生了数据流图。面向数据流的设计能方便地将 DFD 转换成程序结构图，DFD 中从系统的输入数据到系统的输出数据流的一连串连续变换将形成一条信息流。DFD 的信息流大体可分为两种类型，一种是变换流，另一种是事务流。

SD 方法的设计步骤有：①复查并精化数据流图；②确定 DFD 的信息流类型；③根据信息流类型分别将变换流或事务流转换成程序结构图；④根据软件设计的原则对程序结构图作为改进。

（3）结构化程序设计（SP）

结构化程序设计（SP）采用自顶向下逐步求精的设计方法和单入口单出口的控制结构。自顶向下逐步求精的设计方法符合抽象和分解的原则， 人们解决复杂问

题时常用的方法。SA 方法和 SD 方法也采用了自顶向下逐步求精的方法，在详细设计时也同样如此。在设计一个模块的实现算法时，先考虑整体后考虑局部，先抽象后具体，通地逐步细化，最后得到详细的实现算法。单入口单出口的控制结构，使程序的静态和动态结构执行过程一致，使程序具有良好的结构。

2.4.1.4 面向数据结构的设计方法

这类方法以数据结构作为设计基础，根据输入/输出数据结构导出程序的结构。Jackson 方法是一种典型的面向数据结构的设计方法。尽管程序中实际使用中的数据结构有许多种，但这些数据结构中数据元素间的逻辑关系只有顺序、选择和重复 3 类。Jackson 方法的设计步骤为：

（1）分析并确定输入和输出数据的逻辑结构，并用 Jackson 图表示；

（2）找出输入数据结构与输出数据结构间有对应关系的数据单元；

（3）从描述数据结构的 Jackson 图导出描述程序结构的 Jackson 图。

2.4.1.5 软件设计的原则

（1）抽象的原则

软件工程中从软件定义到软件开发要发经历多个阶段，在这个过程中每前进一步都可看作是对软件设计的抽象层次的一次细化。抽象的最低层次就是实现该软件的源程序代码。在进行曲模块化设计时也可以有多个抽象层次，最高抽象层次的模块用概括的方式叙述题的解法，较低抽象层次的模块是对较高抽象层次模块问题解法描述的细化。过程抽象和数据抽象是常用的两种主要抽象手段。

（2）模块化的原则

模块化是指将一个待开发的软件分解或成若干个小的简单的部分模块，每个模块可独立地开发、测试，最后组装成完整的软件。

（3）信息隐蔽的原则

信息隐蔽是开发整体程序结构时使用的法则，即将每个程序的成分隐蔽或封装在一个单一的设计模块中，定义每一人模块时尽可能少地显露其内部的处理。信息隐蔽原则对提高软件的可修改性、可测试性和可移植性都有重要的作用。

（4）模块独立的原则

模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系比较简单。衡量模块独立程度标准有两个：耦合和内聚，耦合是指模块之间联系的紧密程度，耦合度越高，则模块的独立性越差。内聚是指模块内部各元素之间的紧密程度，耦合度越高，则模块的独立性越差。内聚是指模块内部各元素

之间联系的紧密程度，内聚度越低，模块的独立性越差。模块独立要求每个模块都是高内聚低耦合的。

2.4.16 编码

编码阶段的任务就是根据详细的设计说明书编写程序。要编写高质量的程序，应注意选择合适的程序设计语言，明确源程序的质量要求，养成良好的程序设计格。

2.4.1.7 软件测试

软件测试的工作量约占软件开发总工作量的 40%以上，其目的是尽可能多地发现软件产品（主要是指程序）中的错误和缺陷。

测试的关键是测试用例的设计，设计方法可分成两类：白盒测试和黑盒测试。白盒测试把程序看成是装在一只透明的盒子里，测试者完全了解程序的结构和处理过程。白盒测试根据程序的内部逻辑来设计测试用例，检查程序中的逻辑通路是否都按预定的要求正确地工作。黑盒测试把程序看成是装在一只不透明的盒子里，测试者完全不了解（或不考虑）程序的结构和处理过程。黑盒测试根据规格说明书规定的功能来设计测试用例，检查程序的功能是否符合规格说明的要求。

软件测试的主要步骤有单元测试、集成测试和确认测试。单元测试也称模块测试，通常单元测试可放在编码阶段，主要用来发现编码和详细设计中产生的错误，一般采用白盒测试。集成测试也租组装测试，它是对由各模块组装而成的模块进行测试，主要检查模块间的接口和通信。集成测试主要用来发现设计阶段产生的错误，通常采用黑盒测试。确认测试的任务是检查软件的功能、性能和其他特征是否与用户的需求一致，它是以需求规格说明书作为依据的测试，通常采用黑盒测度。

大多数软件生产者使用一种 Alpha 测试和 Beta 测试的过程，来揭露仅由最终用户才能发现的错误。 Alpha 测试是在开发者的现场由客户来实施的，被测试的软件是在开发者从用户的角度进行常规设置的环境下运行的。Beta 测试是在一个或多个客户的现场由该软件的最终用户实施的。与 Alpaha 测试不同的是，进行 Beta 测试时开发者通常是不在场的。

2.4.1. 8 面向对象方法的基本概念

面向对象（00）方法成为软件开发的一种主要方法。它有几个基本概念。

（1）对象

在计算机系统中，对象是指一组属性以及这组属性上的专用操作的封装体。属性可以是一些数据，也可以是另一个对象。每个对象都有它自己的属性值，表示该对象的状态，用户只能看见对象封装界面上的信息，对象的内部实现对用户是隐蔽的。封装目的是使对象的使用者和生产者分离，使对象的定义和实现分开。一个对象通常可由对象名、属性和操作这 3 个部分组成。

（2）类

类是一组具有相同属性和相同操作的对象的集合。一个类中的每个对象都是这个类的一个实例（Instance ）。在分析和设计时，我们通常把注意力集中在类上，而不是具体的对象上。通常把一个类和这个类的所有对象称为类及对象或对象类。

（3） 继承

继承是在某个类的层次关联中不同的类共享属性和操作的一种机制。一个父类可以有多个子类，这些子类都是父类的特例。父类描述了这些子类的公共属性的操作，子类中还可以定义它自己的属性和操作。一个子类只有唯一的一个父类，这种继承称为单一继承。一个了类有多个父类，可以从多个父类中继承特性，这种继承称为重多继承。

（4）消息

消息的对象间通信的手段、一个对象通过向另一对象发送消息来请求其服务。一个消息通常包括接收对象名、调用的操作名和适当的参数（如有必要）。消息只告诉接收对象需要完成什么操作，但并不能指示接收者息样完成操作。消息完全同接收者解释，接收者独立决定采用什么方法来完成所需的操作。

（5）多态性和动态绑定

多态性是指同一个操作作用不同的对象可以有不同的解释，产生不同的执行结果。

与多态性密切相关的一个概念就是动态绑定。传统的程序设计语言把过程调用与目标代码的连接放在程序运行前进行，称为静态绑定。而动态绑定则是把这种连接推迟支动行时才进行。在运行过程中，当一个对象发送消息请求服务时，要根据接收对象的具体情况将请求的操作与实现的方法连接，即动态绑定。

2.4.1.9 面向对象的分析与设计方法

（1）Peter Coad 和 Edward Yourdon 的 OOA 和 OOD 方法

OOA（面向对象的分析）模型由 5 个层次（主题层、对象类层、结构层、属性层和服务层）和 5 个活动（标识对象类、标识结构、定义主题、定义属性和定义服务）组成。在这种方法中定义了两种对象类之间的结构，一种称为分类结构，一种称为组装结构。分类结构就是所谓的一般与特殊的关系。组装结构则反映了对象之间的整体与部分的关系。

OOA 在定义属性的同时，要识别实例连接。实例连接是一个实例与另一个实例的映射关系。

OOA 在定义服务的同时要识别消息连接。当一个对象需要向另一对象发送消息时，它们之间就存在消息连接。

OOA 中的 5 个层次和 5 个活动继续贯穿在 OOD（画向对象的设计）过程中。OOD 模型由 4 个部分组成。它们分别是设计问题域部分、设计人机交互部分、设计任务管理部分和设计数据管理部分 。

（2）Booch 的 OOD 方法

Booch 认为软件开发是一个螺旋上升的过程。在螺旋上升的每个周期中，有 4 个步骤：标识类和对象、确定它们的含义、标识它们之间的关系、说明每一个类的界面和实现。

（3）OMT 方法

对象建模技术 OMT 定义了 3 种模型，它们是对象模型、动态模型和功能模型，OMT 用这 3 种模型来描述系统。OMT 方法有 4 个步骤：分析、系统设计、对象设计和实现。OMT 方法的每一个步骤都使用这 3 种模型，每一个步骤对这 3 种模型不断地进行细化和扩充。

对象模型描述系统包括对象的静态结构、对象之间的关系、对象的属性和对象的操作。OMT 的对象模型中除了对象、类和继承外，还有链、关联、泛化、聚合和模块等概念。

动态模型用来描述与值的变换有关的系统特征--功能、映射、约束和函数依赖。功能模型用数据流图来表示。

OMT 主要步骤：

①分析是 OMT 方法的第一步，其目的是建立可理解现实世界模型。

②在系统设计阶段将确定整个系统的体系结构，以形成求解问题和建立解答的高层次策略。

③对象设计。在分析的基础上，对象设计阶段将建立基于分析模型的设计模型，并考虑实现的细节。设计人员会根据系统设计期间建立的策略把实现细节加入到设计模型中。

④实现阶段将对象设计阶段开发的对象类及基关系转换成特定的程序设计语言、数据库或硬件实现。

2.4.1.10 软件质量保证

软件质量保证是指为保证软件系统或软件产品最大限度地满足用户要求而进行的有计划、有组织的活动，其目的是生产高质量的软件。有多种软件质量模型来

描述软件质量特性，著名的有 ISO/IEC 9126 软件质量模型和 Mc Call 软件质量模型。

软件质量保证环节包括的主要工作有：应用技术方法、进行正规的技术评审、测试软件、标准的实施、控制变动、度量、记录保存和报告。

2．4．1．11 软件开发工具与环境

用来辅助软件开发、运行、维护、管理和支持等过程中的活动的软件称为软件工具，通常也称为 CASE 工具。软件工具大都包含了检测机制，能及时发现一些错误，对提高软件的 质量起着重要的作用。

软件开发环境则把一组相关的工具集成在环境中，环境机制提供数据集成、控制集成和界面集成等机制。数据集成机制为工具提供统一的数据接日；控制集成机制实现工具间的通信和协同工作；界面集成机制使这些工具具有统一的界面风格，从而为软件开发、维护、管理等过程中的各项活动提供连续的、一致的全方位支持。

2．4．2 试题解析

对高级程序员级考试来说，软件工程的重要性非常突出，从历年试题统计（见表 2—6）来看，在各大知识点中平均所占的比重最大，平均每年有 2 道题，这也说明对高级程序员的软件分析与设计能力的要求比较高。总的来说，软件设计方法、软件测试方法、软件模块划分、软件质量要求等是反复考查的重点内容。软件管理、软件工具、软件开发环境等从未考过的内容也应引起重视。

表 2-6 历年程序员级数据结构基础试题统计

试题	考查知识点
1990 年试题 1	可移植性（属于软件质量要求）
1990 年试题 5	软件模块划分，涉及模块内聚
1991 年试题 1	软件测试
1991 年试题 5	软件模块划分，涉及模块耦合
1992 年试题 1	软件质量要求
1992 年试题 3	Jackson 结构化程序设计方法（面向数据结构）
1992 年试题 5	软件测试
1993 年试题 4	结构化设计方法
1993 年试题 5	软件工程标准
1994 年试题 5	软件文档编制
1994 年试题 6	结构测试
1995 年试题 4	软件维护，涉及可维护性
1995 年试题 6	软件测试



高级程序员考试试题分类精解

续表

试题	考查知识点
1996 年试题 2	常用的软件设计方法
1996 年试题 7	软件测试，涉及逻辑覆盖
1997 年试题 6	软件测试用例设计，涉及黑盒测试和因果图方法
1998 年试题 1	软件质量要求
1999 年试题 5	软件模块划分，涉及模块耦合与模块内聚
1999 年试题 6	OMT（对象建模技术）属于面向对象方法
2000 年试题 5	数据流图
2000 年试题 6	软件质量要求与软件质量特性

试题 1 （2000 年试题 5）

从供选择的答案中，选出应填人下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

在软件开发过程中常用图作为描述工具。如 DFD 就是面向 A 分析方法的描述工具。在一套分层 DFD 中，如果某一张图中有 N 个加工（Process),则这张图允许有 B 张子图。在一张 DFD 图中，任意两个加工之间 C。在画分层 DFD 时，应注意保持 D 之间的平衡。DFD 中从系统的输入流到系统的输出流的一连串连续交换形成一种信息流，这种信息流可分为 E 两类。

供选择的答案

A：①数据结构 ②数据流 ③对象 ④构件（ComPonent）

B：①0 ②1 ③1-N ④0-N

C：①有且仅有一条数据流

②至少有一条数据流

③可以有 0 条或多条名字互不相同的数据流

④可以有 0 或多条数据流，但允许其中有若干条名字相同的数据流

D：①父图与其子图②同一父图的所有子图
③不同父图的所有子图④同一子图的所有直接父图

E：①控制流和变换流②变换流和事务流
③事务流和事件流④事件流和控制流

【解析】

本题考查数据流图（DFD）的基本知识。

在软件需求分析阶段，用 SA 方法产生了数据流图。数据流图是结构化分析方法的一种分析结果，用来描述数据流从输入到输出的变换过程。数据流图的基本成份有数据流、加工。文件和源 / 宿。

一个软件系统，其数据流图往往有多层。如果父图有 N 个加工，则该父图可以有 0～N 张子图，但是每张子图只能孙对应于一张父图。

在画数据流图时，应注意父图与子图的平衡，即父图中某加工的输入输出数据流必须与
其子图的输入输出流在数量和名字上相同。

DFD 的信息流大体可分为两种类型，一种是变换流，另一种是事务流。

［答案］A：②B：④C：③D：①E：②

试题 2 （2000 年试题 6）
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

软件的易维护性是指理解、改正、改进软件的难易程度。通常影响软件易维护性的因素有易理解性、易修改性和 A 。在软件的开发过程中往往采取各种措施来提高软件的易维护性。如采用 B 有助于提高软件的易理解性；C 有助于提高软件的易修改性。

在软件质量特性中，D 是指在规定的一段时间和条件下，与软件维持其性能水平的能力有关的一组属性；E 是指防止对程序及数据的非授权访问的能力。

供选择的答案

A：①易使用性②易恢复性③易替换性④易测试性

B：①增强健壮性②信息隐蔽原则
③良好的编程风格④高效的算法

C：①高效的算法②信息隐蔽原则③增强健壮性④身份认证

D：①正确性②准确性③可靠性④易使用性

E：①安全性②适应性③灵活性④容错性

【解析】

本题考软件质量要求和软件质量特性。

对于源程序的质量要求，最基本的就是正确性和可靠性，除此之外，更要注重程序的易使用性、易维护性和易移植性。易使用性就是要对用户友好，便于用户使

用，做到少培训甚至零培训；易维护性包括易理解性、易测试性和易修改性；移植性则指程序从某一环境移植到另一环境的能力。信息隐蔽是开发整体程序结构时使用的法则，即将每个程序的成分隐蔽或封装在一个单一的设计模块中，定义每一个模块时尽可能少地显露其内部的处理。信息隐蔽原则对提高软件的可修改性、可测试性和可移植性都有重要的作用。

一般使用 ISOflEC 9126 软件质量模型来描述软件质量特性，它有 3 个层次组成：第 1 层是质量特性；第 2 层是质量于特性；第 3 层是度量指标。题中提到的可靠性位于第 1 层，安全性位于第 2 层。

[答案]A：④ B：③ C：② D：③ E：①

试题 3（1999 年试题 5）
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

软件设计中划分模块的一个准则是 A 。两个模块之间的耦合方式中，B 耦合的耦合度最高，C 耦合的耦合度最低。一个模块内部的内聚种类中 D 内聚的内聚度最高，E 内聚的内聚度最低。

供选择的答案

A：①低内聚低耦②低内聚高耦合③高内聚低耦合④高内聚高耦合

B：①数据②非直接③控制④内容

C：①数据②非直接③控制④内容

D：①偶然②逻辑③功能④过程

E：①偶然②逻辑③功能④过程

【解析]

软件设计中划分模块的一个准则是追求更高的内聚度和更低的耦合度。耦合度是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接日的复杂程度、进入访问一个模块的点以及通过接口的数据。在软件设计中应该追求尽可能松散的耦合系统，在这样的系统中可以研究、测试或修改、维护任何一个模块，而不需要对系统的其他模块有很多了解或影响其他模块的实现。此外，当某处发生错误时，低耦合度系统的错误传播的范围相应小些。因此，模块间的耦合程度直接影响系统的可理解性、可测试性、可靠性和可维护性。

如果两个模块中的每一个都能够独立地工作而不需要另一个模块的存在，那么他们之间就没有耦合关系；如果两个模块彼此通过参数交换数据，而交换的信息仅仅是数据，那么这种耦合称为数据耦合；如果传递的信息中也有控制信息，则这种耦合称为控制耦合。

当两个或多个模块通过公共数据环境相互作用时，它们之间的耦合称为公共耦合；如果两个模块之间有下列情况之一，则称两个模块之间的耦合关系为公共耦合。（1）一个模块访问另一个模块的内部数据；（2）一个模块没有通过正常人口而转到另一个模块内部；（3）两个模块有一部分程序代码重叠；（4）一个模块有多个入口。

数据耦合的耦合程度最低，控制耦合次之，适当分解模块可以用数据耦合代替控制耦合。公共耦合的复杂程度随耦合模块的个数和耦合模块对公共数据环境的操作而变化。内容耦合是程度最高的耦合，在程序中应该坚决避免内容耦合。

内聚标志着一个模块内各个元素彼此结合的紧密程度，它是信息隐蔽和局部化概念的自然扩展。简单地说，理想的内聚模块只做一件事情。

如果一个模块完成一组任务，这些任务彼此间即使有关系，也是松散的关系，就叫做偶然内聚；如果一个模块完成的任务在逻辑上属于相同或相似的操作，则称为逻辑内聚；如果一个模块包含的任务必须在同一段时间内执行，就叫做时间内聚。

如果一个模块内的处理元素是相关的。而且必须以特定次序执行，则称为过程内聚；如果模块中所有的元素都使用同一个输入数据和（或）产生同一个输出数据，则称为通信内聚。

如果一个模块内的处理元素和同一个功能密切相关，而 B 这些外排必须顺序执行．则称为顺序内聚；如果模块内所在处理元素属于一个整体，完成一个单一的功能，则称为功能内 聚。

内聚程度从低到高的排列是：偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚。顺序内聚、功能内聚。

【答案】A：③B：④C：②D：③E：①

试题 4 （1999 年试题 6）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

OMT 是一种对象建模技术，它定义了 3 种模型，它们分别是 A 模型、B 模型和 C 模型。其中， A 模型描述了系统中对象的静态结构，以及对象之间的联系，B 模型描述系统中与时间和操作顺序有关的系统特征，表示瞬时的行为上的系统的“控制”特征，通常可用 D 来表示，C 模型描述了与值的变换有关的系统特征，通常可用 E 来表示。

供选择的答案

A：①对象②功能③ER④静态

B：①控制②时序③动态④实时

C：①对象②功能③变换④计算

D：①类图②状态图③对象图④数据流图

E：①类图②状态图③对象图④数据流图

【解析】

OMT（对象建模技术）是一种围绕真实世界的概念来组织模型的软件开发方法。OMT 从问题陈述开始，理解问题陈述中的客观世界，将其本质抽象成模型表示，建立系统的 3 种模型，即对象模型、动态模型和功能模型。

对象模型描述了系统中对象的静态结构以及对象间的联系，用对象模型图来表示。对象模型图是 ER 图的一种拓广形式。动态模型描述了与时间和操作次序有关的系统属性，动态模型由多张状态图组成。各个类的状态图通过共享事件组成系统的动态模型。功能模型描述系统内数据值的变化，它由数据流图组成。数据流图说明数据流是如何从外部输入、经过操作和内部存储而到外部输出的。OMT 的 3 种模型相辅相成，组成系统的一个完整的正交视图。

OMT 方法体现了面向对象的系统开发方法的基本特点，强调对系统结构的理解，而不是系统功能的分解。在 OMT 方法中，对象模型最重要，动态模型次之，最后是功能模型。

使用面向对象的基本思想构造的系统模型与客观系统的结构十分类似，可以使用该模型与用户通信。

在分析阶段产生系统模型后，系统设计的任务主要是细化模型。分析和设计可使用统一表示方法，省略了类似结构方法中从数据流图到模块调用层次的转换过程，而且这种模型用面向对象的程序设计语言来实现也显得十分自然。

相对于传统的软件工程方模学，OMT 的开发重心转移到了分析阶段，使得分析的结果比一般的软件开发方法更为可靠，减少了因分析不透彻引起的问题。OMT 支持系统的无缝开发。在整个开发过程中使用统一的软件概念即对象，所有其他概念都是围绕对象组成的，在分析阶段开发的对象模型也适用于设计和实现阶段。这样，软件开发的阶段性就不那么明显了。由于各阶段是一致吻合的，很容易实现各阶段的反复，而且每一次反复都是对系统的进一步深化。

但 OMT 方法还存在以下几个方面的不足：（1）对问题陈述论述得不足；（2）3 种模型的一致性难以检测和维护，而且对系统约束的描述能力也不足：（3）建模过程描述得不很清晰，建模的结果即各种图形表示也不能体现建模的过程，从而增加了分析人员掌握该方法的
难度；（4）OMT 方法使用的图形在分析大系统时显得条理不够清晰。

［答案］A：①B：③C：②D：②E：④

试题 5（1998 年试题 1）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

设计高质量的软件是软件设计追求的一个重要目标。可移植性、可维护性、可靠性、效率、可理解性和可使用性等都是评价软件质量的重要方面。

可移植性反映出把一个原先在某种硬件或软件环境下正常运行的软件移植到另一个硬件或软件环境下，使该软件也能正确地运行的难易程度。为了提高软件的可移植性，应注意 提高软件的 A 。

可维护性通常包括 B 。通常认为，软件维护工作包括改正性维护、C 维护和 D 维护。其中 C 维护则是为了扩充软件的功能或提高原有软件的性能而进行的维护活动。E 是指当系统万一遇到未预料的情况时，能够按照预定的方式做合适的处理。

供选择的答案

A：①使用方便性②简洁性③可靠性④设备不依赖性

B：①可用性和可理解性②可修改性、数据独立性和数据一致性
③可测试性和稳定性④可理解性、可修改性和可测试性

C、D：①功能性②扩展性③合理性④完善性
⑤合法性③适应性

E：①可用性②正确性③稳定性④健壮性

【解析］

软件的可移植性是指把程序从一种硬件配置或软件系统环境转移到另一种配置和环境时，需要的工作量的多少。提高软件可移植性的关键在于提高软件的设备无关性，即设备不依赖展性。

软件的可维护性通常包括叮理解性、可修改性和可测试性。按照每次维护的具体目标，软件维护工作可分为 3 类：改正性维护、完善性维护和适应性维护。

改正性维护的目的在于纠正开发期间未能发现的错误。由于软件测试的不彻底性，任何大型软件在交付使用后都会发现一些潜藏的错误，对它们进行的诊断和改正就称为改正性维 护，改正性维护约占总维护的 20％。

完善性维护指的是任何软件，无论是应用软件还是系统软件，在使用期间都要不断改善，加强产品的功能和性能，以满足用户日益增长的需求，提高自己产品的

市场竞争力。刚投入使用的是第 1 版，以后就可能不断升级为第 2 版、第 3 版等，在整个维护工作量中，完善性维护所占比重最大，约占 50～60％。

适应性维护是指软件为适应运行环境的变化而进行的一种维护，如硬件或支撑软件（如操作系统升级）改变引起的变化，将软件移植到其他的运行平台上等，这类维护大约占总维量的 25％。

软件的健壮性是指在硬件发生故障、输入的数据无效或操作错误等意外环境下，即系统遇到未预料的情况时，系统能够做出适当响应的程度。

［答案】A：④B：④C：④D：③E：④

试题 6 （1997 年试题 6）
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

在设计测试用例时，A 是用得最多的一种黑盒测试方法。在黑盒测试方法中，等价类划分方法设计测试用例的步骤是：

1．根据输入条件把数目极多的输入数据划分成若干个有效等价类和若干个无效等价类；

2．设计一个测试用例，使其覆盖 B 尚未被覆盖的有效等价类，重复这一步，直至所有的有效等价类均被覆盖。

3．设计一个测试用例，使其覆盖 C 尚未被覆盖的无效等价类，重复这一步，直至所有的无效等价类均被覆盖。

因果图方法是根据 D 之间的因果关系来设计测试用例的。

在实际应用中，一旦纠正了程序中的错误后，还应选择部分或全部原先已测试过的测试用例，对修改后的程序重新测试，这种测试称为 E 。

供选择的答案

A：①等价类划分②边值分析③因果图④判定表

B、C：①1 个②7 个左右③一半④尽可能少的
⑤尽可能多的③全部

D：①输入与输出②设计与实现③条件与结果④主程序与子程序

E：①验收测试②强度测试③系统测试④回归测试

[解析]

等价类划分是典型的黑盒测试方法，其方法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据当作测试用例。列举所有可能的测试数据进行测试工作量太大，通常选取一部分测试数据进行测试，因此如何选取数据就成为关键的一步。用这种方法设计测试用例需要经过划分等价类、选取测试用例两个步骤。

划分等价类时，首先把数目极多的输入分成若干个等价类。所谓等价类就是某个输入域的集合，对于一个等价共中的输入值来说，它们揭示程序中错误的作用是等效的。如果我们的测试用例全部从一个等价类中选取，测试工作将不能保证软件质量。

根据列出的等价类表，应该按照以下步骤确定测试用例：

（1）为每个等价类规定一个唯一的编号；

（2）设计一个等价类，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，最后
使得所有有效等价类都被测试用例所覆盖；

（3）设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步使所有无效等价
类都被覆盖。

应当注意到，（3）中规定一次只能覆盖一个无效等价类，因为一个测试用例中如果含有多个错误，有可能在测试中只发现其中的一个，另一些被忽视。

因果图法是根据输入与输出之间的因果关系来设计测试用例的，要检查输入条件的各种组合情况，在设计测试用例时，需分析规格说明中哪些是原因，哪些是结果，并指出原因和结果之间、原因和原因之间的对应关系。因果图法最终生成的是判定表；功能图方法利用功能图形式化地表示程序的功能说明，并机械地生成功能图的测试用例；回归测试是在纠正了程序中的错误后，选择部分或全部原先已测试过的测试用例，对修改后的程序重新测试，以 验证对软件修改后有没有引出新的错误；强度测试是检查在系统运行环境发生故障的情况下，系统可以运行到何种程度的测试；系统测试是将通过确认测试的软件作为整个基于计算机系统的一个元素，与计算机硬件、外设、支持软件、数据以及人员等其他系统元素结合在一起，在实际运行环境对计算机系统进行一系列的组装测试和确认测试；验收测试是在系统进行有效性测试及软件配置审查后，以用户为主进行的测试。

【答案】A：②B：⑤C：①D：①E：④

试题 7 （1996 年试题 2）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

在软件工程的设计阶段中，有 3 种常用的设计方法：结构化设计（SD）方法、Jackson 方法和 Parnas 方法。SD 方法侧重于 A ；Jackson 方法则是 B ；Pmas 方法的主要思想 C 。从 70 年代中期到 90 年代早期，D 是最为常用的设计方法。E 法只提供了重要的设计准则，没有规定出具体的工作步骤。

供选择的答案

A～C：

①使用对象、类和继。②由数据结构导出模块结构。

③模块要相对独立，且功能单一，使块间联系弱，块内联系强。

④将可能引起变化的因素隐藏在某有关模块内部，使这些因素变化时的影响范围受到限制。

⑤用数据流图表示系统的分解，且用数据词典和小说明分别表示数据和加工的含义。

⑥自顶向下、逐步细化，采用顺序、选择和循环 3 种基本结构，以及限制 got0 语句的使用，设计出可靠的和易维护的软件。

D：①SD②Jackson③Parnas④面向对象

E：①SD②Jackson③Parnas④以上皆非。

【解析】

结构化设计（SD）方法、Jackson 方法和 Parnas 方法是软件工程设计阶段常用的设计方法。

结构化设计（SD）采用自顶向下逐步求精、模块化的设计方法，单入日、单出日的控制结构，利用程序结构图表达模块之间的关系。可见结构化设计方法是以模块化设计为中心，在开始设计时就把待开发系统划分为若干个相互独立的模块，每一个模块的工作明确清晰，模块之间的耦合量低，可减少修改或重新设计时的工作量。结构化设计方法的关键是要恰到好处地划分模块，采用试探方法处理好模块内部以及模块之间的联系，从而达到逐步疏清条理的目的。所以问题 A 选择③。结构化设计方法是 70 年代中期到 90 年代早期最常用的设计方法，问题 D 选择①。

Jackson 方法是一种面向数据结构的设计方法，设计目标是得出对程序处理过程的描述， 其设计过程是从描绘数据结构的 Jackson 图推导出描绘程序结构的 Jackson 图，该方法适用于比较简单的数据处理系统，所以问题 B 选择②。

Parnas 方法强调在概要设计时应预先估计在未来生存周期中可能发生的情况，并采取相应措施来提高系统的可维护性和可靠性。信息隐藏是提高软件可维护性

的重要措施，在分解模块时，就应采取措施，将一些将来可能发生变化的因素隐含在某模块内，使将来因修改造成的影响尽可能地局限在一个或少数几个模块中，这种方法只提供了重要的设计准则，而没有规定具体的工作步骤，所以问题C选择④，问题E选择③。

［答案］A：③ B：② C：④ D：① E：③

试题 8 （1996年试题 7）
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

软件测试的目的是 A 。通常 B 是在代码编写阶段可进行的测试，它是整个测试工作的基础。

逻辑覆盖标准主要用于 C 。它主要包括条件覆盖、条件组合（多重条件）覆盖。判定覆盖、条件及判定覆盖、语句覆盖和路径覆盖等几种，其中除路径覆盖外最弱的覆盖标准是 D ，最强的覆盖标准是 E 。

供选择的答案

A：①表明软件的正确性②评价软件质量
③尽可能发现软件中错误④判定软件是否合格

B：①系统测试②安装测试③验收测试④单元测试

C：①黑盒测试方法②白盒测试方法③灰盒测试方法④软件验证方法

D、E：①条件覆盖②条件组合覆盖③判定覆盖
④条件及判定覆盖⑤语句覆盖

【解析】

软件测试在软件生存周期中占有重要地位，这不仅是因为测试阶段占用的时间、花费人力和成本占软件开发比重的40%以上，而且还因为它是保证软件质量的关键步骤。

由于人的主观因素或客观原因，在软件开发过程中不可避免的要产生一些错误。软件测 试的任务是在软件投入运行以前尽可能多地发现并改正软件中的错误，所以问题A应选择③。

一个软件产品在交付使用前要经历以下几种测试：

（1）模块测试。也称为单元测试，是针对每个模块单独进行的测试。模块测试一般和程序编写结合起来，在编码阶段由软件编写者进行测试，以保证每个模块作为一个单元能正确运行，所以问题B选择④。

（2）于系统测试。把已通过单元测试的模块连接成为子系统来测试，着重检验模块间的接口。

（3）系统测试。把经过测试的子系统连接成为一个完整的系统米测试，主要检查系统是 否实现说明书中指定的功能，系统的动态性能是否符合要求。

（4）验收测试。是为了确认已开发的软件能否满足标准，是否合格。与系统测试不同的是，它是在客户的积极参与下进行的。

（5）平行运行。一些十分重要的软件在经过验收之后，并不立即投入运行，而是再经过一段平行运行的测试，即新旧两个系统同时运行，比较结果有什么不同。

下面再对题目中涉及的一些概念进行解释。

黑盒测试方法又称功能测试，把程序看作一个黑盒子，在完全不考虑程序内部结构的情况下设计测试数据，主要测试程序的功能是否符合软件说明书的要求。

白盒测试方法又称结构测试，它是根据程序的内部结构设计测试数据，检查程序中的每条通路是否都能按要求正确运行。

逻辑覆盖主要用于白盒测试方法，所以问题 C 选择②。由于覆盖的详尽程度不同，又分为语句覆盖、判定覆盖、条件覆盖、条件组合覆盖、条件及判定覆盖和路径覆盖等。

语句覆盖：设计足够多的测试用例，使程序中的每个语句至少执行一次。语句覆盖是最弱的逻辑覆盖准则。

判定覆盖：设计足够多的测试用例，不仅每个语句至少执行一次，而且使得程序中每个判定的每个分支至少执行一次。

条件覆盖：设计足够多的测试用例．不仅每个语句至少执行一枚．而日体每个判定表达式中每个条件都取到可能的结果。

条件组合覆盖：设计足够多的测试用例，使得每个判断的各种可能组合至少出现一次。

条件及判定覆盖：设计足够多的测试用例，使得判断中每个条件的每种可能至少出现一次，而

且每个判断的不同判定结果也至少出现一次。

路径覆盖：设计足够多的测试用例，要求程序中所有路径至少执行一次。

在上述几种逻辑覆盖中，除路径覆盖外，最弱的覆盖标准是语句覆盖，最强的覆盖标准是条件组合覆盖。

【答案】A：③ B：④ C：② D：⑤ E：②

试题 9（1995 年试题 4）

从供选择的答案中，选出应填入下面叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

软件维护工作越来越受到重视，因为它的花费常常要占软件生存周期全部花费的 A%左右。其工作内容为 B，为了减少维护工作的困难，可以考虑采取的措施是 C。而软件的可维护性包 D。所谓维护管理主要指的是 E。

供选择的答案：

A：①10～20 ②20～40 ③60～80 ④90 以上

B：①纠正与修改软件中含有的错误。

②因环境已发生变化，软件需作相应的变更。

③为扩充功能，提高性能而作的变更。

④包括上述各点内容。

C：①设法开发出的无错的软件

②增加维护人员数量。

③切实加强维护管理，并在开发过程中就采取有利于未来维护的措施。

④限制个性的范围。

D：①正确性、灵活性、可移植性。

②可测试性、可理解性、可个性性。

③可靠性、可复用性、可用性。

④灵活性、可靠性、高效性。

E：①加强需求分析。 ②重新编码。
③判定修改的合理性并审查修改质量。 ④加强维护人员管理。

【解析】

软件的可维护性、可用性、可靠性构成了衡量软件技师的几个重要尺度。

软件维护是指已经完成发工作，对软件产品所进行的后续活动。一般来说软件维护活动贯穿了软件投入使用直到软件被淘汰的整伸过程，在这个过程中，维护工作需要解决开发阶段所遇到的各种问题和解决某些维护工作本身的特有问题。

软件维护工作包括 3 个方面：改正性维护、适应性维护和完善性维护。改正性维护是在软件运行中发生异常或故障时进行的，这种故障往往是由于软件开发过程中某个环节上的隐患构成的。适应性维护的目的是要使运行的软件能适应外部环境的变动，例如数据格式的变动、数据输入输出方式的变动等都会影响软件的正常工作。完善性维护则是为扩充软件的功能，提高软件的性能而开展的软件维护活动，如用户在使用了一段时间之后对提出了新的要求，这种情况下，就需要完善性维护。在整个软件维护活动中，完善性维护所占的比重最大，平均在 50%左右。统计资料表明，维护阶段的花费占整个软件生存周期花费的 67%，这是一个相当可观的数字。

正确的软件维护工作所应采取的措施是：切实加强维护管理，并在开发过程中就采取有利于软件未来维护的措施。软件的维护不仅是技术性的，而且还需要大量的管理工作与之配合。从原则上讲，维护工作从理解软件开始，在些基础上，如果有明确的维护任务，则针对些任务提出修改建议；在经过部门的审批之后，正确的维护建议被批准；按照修改方案进行的修改结束后，为保证修改的质量，应该进行严格的测试；经过管理部门再次审查后，教授可以对文档进行正式修改。这样，所谓维护管理主要是指判定个性的合理过性并审查修改质量。

【答案】A： ③ B： ④ C： ③ D： ② E： ③

试题 10（1995 年试题 6）

从供选择的答案中，选出应填入下面叙述中{ }内的正确答案，把编号写在答卷的对应栏内。
软件测试是软件质量保证的主要手段之一，测试的费用已超过 A 的 30%以上。因此提高测试的有效性非常重要。"高产"的测试是指 B。根据国家标准 GB8566-88 计算机软件开发规范的规定，软件的开发和维护分为 8 个阶段，其中单元测试是在 C 阶段完成的；组装测试的计划内是在 D 阶段制定的；确认测试的计划是在 E 阶段制定的。

供选择的答案

A：①软件开发费用。 ②软件维护费用。
③；软件开发和维护费用。 ④软件研制费用。

B：①用适量的测试用例，说明被测程序正确无误。
②用适量的测试用例，说明被测程序符合相应的要求。
③用少量的测试用例，发现被测程序尺可能多的错误。
④用少量的测试用例，纠正被测程序尺可能多的错误。

C～E：①可行性研究和计划。②需求分析。③概要设计。④详细设计。
⑤实现。 ⑥组装测试。⑦确认测试。⑧使用和维护。

【解析】

目前，在大中型软件开发项目中，测试都占据着重要地位，同时，测试也是在将软件交付给客户之前所必须完成的步骤。测试所花费用已超过软件开发费用的30%以上。如何组织好测试，特别是如何选择测试用例，对保障软件质量，降低测试费用有着重要的意义。

一个高效的测试，是指通过对所设计的少量测试用例进行测试，从而发现被测试程序中尺可能我的问题，并完成修改。测试按照被测试的内容可分为 3 种：单元测试(对程序单元或模块单独进行测试)、组装测试(把已通过单元测试的模块连接起来，测试模块间的接口及软件设计中的问题，常用功能测试办法)和确认测试(对软件技师作全面测试，以确认开发的软件是否符合验收标准)。

根据国家标准 GB8566-88 计算机软件开发规范的规定，软件开发和维护分为 8 个阶段，分别为可行性研究和计划、需求分析、概要设计、详细设计、实现、组装测试、确认测试和使用维护。GB8566-88 规定单元测试在实现阶段完成；组装测试在组装测试阶段完成，但组装测试的计划应该在概要设计阶段制订，而确认测试的计划则在需求分析阶段就应该制订好。

【答案】A：① B：③ C：⑤ D：③ E：②

试题 11 (1994 年试题 5)

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

国家标准《计算机软件产品开发文件编制指南 GB8567-88》中规定，在一项软件开发过程， 一般地说应该产生 14 种文件，其中管理人员主要使用的有 A、B、C、开发进度月报、项目开发总结报告。开发人员主要使用的有 A、B、D、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划和 E。维护人员主要使用的有设计说明书、E 和 C。

A～E： ①软件需求说明书 ②项目开发计划 ③可行性研究报告
④模块开发卷宗 ⑤测试分析报告 ⑥操作手册 ⑦用户手册

【解析】

《计算机软件产品开发文件编制指击 GB8567-88》是《计算机软件开发规范 B8566-88》的配
套文件，由国家标准局在 1988 年 1 月批准并发布，规范中详细规定了软件开发过程中各个阶段及每一阶段的任务、实施步骤、实施要求、完成标志及交付的文本。《计算软件产吕开发文件编制指击 GB567-88》则为应交付的文本提供了编写指南，一般来说，规定在一项软件开

发过程中应该产生 14 种文件：

A 可行性研究报告

B 项目开发计划

C 软件需求说明书

D 数据要求说明书

E 概要设计说明书

F 详细设计说明书

G 数据库设计说明书

H 用户手册

I 操作手册

J 模块开发卷宗

K 测试计划

L 测试分析报告

M 开发进度月报

N 项目开发总结报告

其中管理人员主要使用的有项目开发计划、可行性研究报告、模块开发卷宗、开发进度月报 和项目开发总结报告；开发人员主要使用的有项目开发计划、可行性研究报告、软件需求说明书、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划和测试分析报告；维护人员主要使用的有设计说明书、测试分析报告和模块开发卷宗。

【答案】A：②B：③C：④D：①E：⑤

试题 12 (1994 年试题 6)

从供选择的答案中，选出应填入{ }内的正确答案，把编号写在答卷的对应内。

在结构测试用例设计中，有语名覆盖、条件覆盖、判定覆盖(即分枝覆盖)、路径覆盖等，其 A 是最强的覆盖准则。为了对图 2-8 所示的程序段进行覆盖测试，必须适当地选取测试数据组。若 x 和 y 是两个变量，可供选择的测试数据组共有Ⅰ、

Ⅱ、Ⅲ、Ⅳ四组(见表 2-7)，则实现判定覆盖至少应采用的测试数据组是 B；实现条件覆盖至少采用的测试数据组是 C，实现路径覆盖至少应采用的测试数据组是 D 或 E。

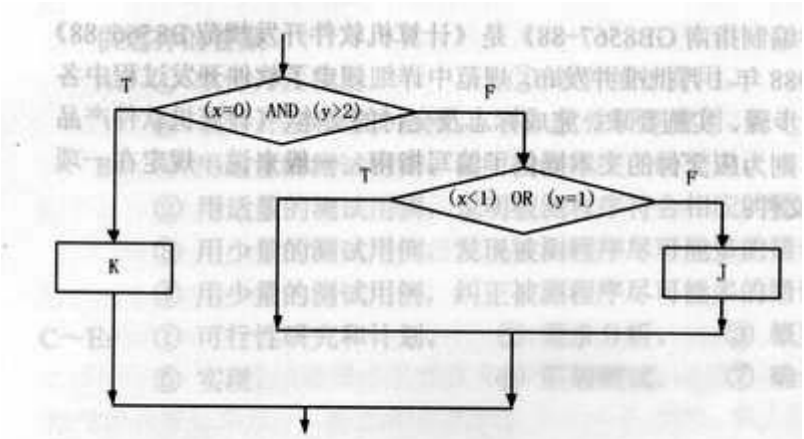


图 2-8 1994 试题 6

表 2-7 1994 试题 6

	x	y
测试数据组Ⅰ	0	3
测试数据组Ⅱ	1	2
测试数据组Ⅲ	-1	2
测试数据组Ⅳ	3	1

供选择的答案

A：①语句覆盖 ②条件覆盖 ③判定覆盖 ④路径覆盖

B～E：①Ⅰ和Ⅱ组 ②Ⅱ和Ⅲ组 ③Ⅲ和Ⅳ组 ④Ⅰ和Ⅳ组⑤Ⅰ、Ⅱ和Ⅲ组 ⑥Ⅱ、Ⅲ和Ⅳ
组 ⑦Ⅰ、Ⅲ和Ⅳ组 Ⅰ、Ⅱ和Ⅳ组

【解析】

测试阶段在软件生命周期中占有重要地位。一个软件产品在交付用户正式使用前主要经过 3 种测试：单元测试、集成测试和验收测试。测试的方法有黑盒法和白盒 黑盒方法主要用于动能测试或数据驱动测试：白盒方法主要用于结构测试或逻辑驱动测试。测试用的数据称为测试用例，正确选取测试用例对测试的最终结

果有很大影响。白盒测试用例的设计方法有语句覆盖、条件覆盖、分支覆盖、路径覆盖等多种，其中语句覆盖是最弱的覆盖准则，路径覆盖则最强。

本题所给出的 4 组测试数据，要实现分支覆盖即判定覆盖，必须使程序中每个逻辑判断的取真分支和取假分支在行某组测试数据时至少历一次，应采用的测试数据组是第Ⅰ和第Ⅱ组数据；要实现条件覆盖，必须使程序中每个条件的可能取值至少满足一次，应采用的测试数据组是Ⅰ和 Ⅱ组数据；要实现路径覆盖，求程序中的所有路径至少经历一次，则应采用的测试数据组是第Ⅰ、Ⅱ、Ⅲ组或者第Ⅰ、Ⅱ、Ⅳ组。

【答案】A：④ B：① C：④ ⑤ E：⑧

试题 13 (1993 年试题 4)

从供选择的答案中，选出应填入下面关于软件设计方法叙述中{}内的正确答案，把编号写在答卷的对应栏内。

结构化设计方法在软件开发中用于 A，它是一种面向 B 的设计方法。该方法使用的图形工具是 C，C 中矩形表示 D。如果两矩形之间有直线相连，表示它们存在 E 的关系。

供选择的答案

A：①测试用例设计 ②概要设计 ③程序设计 ④详细设计

B：①对象 ②数据结构 ③数据流 ④控制流

C：①程序结构图 ②数据流程图 ③程序流程图 ④实体联系图

D：①数据 ②加工 ③模块 ④存储

E：①链接 ②调用 ③并列 ④顺序执行

【解析】

软件设计工作分成概要设计和详细设计两步。结构化设计(Structured Design，简称 SD)方法是结构化分析(SA)的后续阶段，用于软件的概要设计。SD 是一种面向数据流的设计方法，从整个程序的结构出发，突出程序模块，利用程序结构图表达程序模块之间的关系，尤其适用于变化型结构和事务型结构的数据处理系统。

程序结构图是采用结构化设计方法进行软件概要设计的描述手段，以图形的形式描述软件系统的模块组成及模块之间的调用关系，程序结构图可以精炼地表达模块化的设计思想，清晰地表达模块之间的联系。

构成程序结构图的主要成分有模块、调用和数据，结构图中的模块用矩形表示，在矩形框内可标上模块的名字。模块间如有箭头或直线相连，表明它们之间有调用关系。对于两个处在不同位置的模块，把上面的称为调用模块，下面的称为被调用模块。调用简头表示模块调用时模块间的数据传送，简头的方向指明了传送的方向，图中的数据也用适当的名称加以标识。

【答案】A：② B：③ C：① D：③E：②

试题 14 (1993 年试题 5)

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

按制定软件工程标准的不同层次和适用范围，软件工程标准可分为 5 级，A 制定的是国际标准，B 是行业标准。GB1256-89 对程序流程图等作了明确、具体规定，这种标准程序流程图的特点有 C、D 和 E。

供选择的答案

A、B：①IEEE，GJB ②IEEE，ANSI ③ISO，IEC
④ISO，IEEE ⑤IEC，GJB ⑥ANSI，ISO

C～E：①箭头表示数据的传递方向。

②允许自行定义多种特定的图形符号。

③对循环结构规定了一对特定的符号。

④它和 ISO 的有关规定有一些差别。

⑤允许有两个以上出口的判断。

⑥特定方向的流线才用箭头标明流向。

⑦不允许在图形符号外加注标识符和描述符。

【解析】

按制定软件工程标准(或规范)的不同层次和适用范围，软件工程标准可划分成国际标准、国家标准、行业标准、企业或地方规范、项目规范等 5 级。
ISO(International Standards Organization，国际标准化组织)和
IEC(International Electromechanical Commission，国际电工委员会)制定的是国际标准；IEEE 是指美国电气和电子工程师学会，GJB 是指中华人民共和国军用标准，后两者属于行业标准。

国家标准《GB 1256-89》是《信息处理-数据流程图：程序流程图、系统流程图、程序网络图和系统资源图的文件编符号集约定》，该标准对程序流程图作了明确具体的规定。在这种标准流程图中，对循环结构规定了一对特定的符号，即循环界限为去上角矩形表示的上界限和去下角矩形表示的下界限，分别表示循环的开始和循环的结束：判断用菱形表示，它只有一个入口，但可以有若干个可供选择的出口；控制流的流线用直线表示，流线的标准流向是从左到右和从上到下，沿标准流向的流线可以不用箭头指示流向，但对于沿非标准流向的流线方向，应该用箭头指示方向。

【答案】A：③ B：① C：③ D：⑤ E：⑥

试题 15（1992 年试题 1）

从供选择的答案中选出应填入{ }的正确答案，把编号写在答卷的对应栏内。

软件质量包含多方面的内容，A、B、可移植性和可复用性等是较为重要的质量特征。在软件开发中，必须采取有力的措施，以确保软件的质量，这些措施至少应包括 C、D 和 E

供选择答案

A、B ①稳定性 ②可靠性 ③数据一致性
④可维护性 ⑤可行性 ⑥数据独立性

C、D、E： ①在开发初期制定质量保证计划，并在开发中坚持实行。

②开发工作严格按阶段进行，文档工作应在开发完成后集中进行。

③严格执行阶段评审。

④要求用户参与全部开发过程，以监督开发质量。

⑤开发前选定或制定开发标准或开发规范，亲遵照实施。

⑥争取足够的开发经费和开发人力的支持。

【解析】

软件质量涉及多方面的内容，其中较为重要的质量特性有软件的可靠性、可维护性、可移植性和可复用性等。这些特性可为软件的安全使用、进一步扩充、修改、复制和移植打下良好的基础。稳定性、数据一致性、可行性和数据独立等都不是与软件质量直接相关的概念。

在软件开发中，应采取有力的措施确保软件的质量。有些措施在实践证中已证明是行之有效的。如在开发初期制定质量保证计划，并在开发中坚持实行；严格执

行阶段评审，以便及时发现问题；开发前选定或制定开发标准或开发规范，并遵照实施；软件生存期的各阶段都有完整的文档等。

【答案】A：② B：④ C：① D：③ E：⑤

试题 16 (1992 年试题 13)

从供选择的答案中选出应填入{ }的正确答案，把编号写在答卷的对应栏内。
Jackson 结构化程序设计方法是英国 M.Jackson 提出的，它是一种面向 A 的设计方法，主要适用于规模适中的 B 系统的开发，其基本步骤依次是 C、D、E。

供选择的答案

A：①对象 ②数据流 ③数据结构 ④控制结构

B：①数据处理 ②文字处理 ③实时控制 ④科学计算

C、D、E：①建立数据结构 ②列出基本操作 ③建立程序结构
④建立控制结构 ⑤建立对象

【解析】

Jackson 方法由英国的 M.Jackson 提出，Jackson 方法的基本思想同软件设计听 SD 方法是一致的，但它不是面向数据流，而是面向数据结构的。应用该方法的基本步骤依次是：建立系统的数据结构；以数据结构为基础，对应地建立程序结构；列出程序中要用到的各种基本操作，再将这些操作分配程序结构适当的模块中。

Jackson 方法的设计原则是将程序结构和数据结构相对应，该方法特别适用于设计企事业信息管理一类的规模适中的数据处理系统。这些系统处理的数据大多具有层次结构，如文件由记录组成，记录又由数据项组成，所以可以以数据结构为基础，相应地建立模块的层次结构，如处理文件的模块调用处理记录的模块，处理记录的模块又调用处理数据项的模块。

【答案】A：③ B：① C：① D：③ E：②

试题 17 (1992 年试题 5)

从供选择的答案中选出应填入{ }内的正确答案，把编号写在答卷的对应栏内。

A 在实验阶段进行，它所依据的模块功能描述和内部细节以及测试方案应在 B 阶段完成，目的是发现编程错误。

C 所依据的模块说明书和测试方案应在 D 阶段完成，它能发现设计错误。

E 应在模拟的环境中进行强度测试的基础上进行，测试计划应在软件需求分析阶段完成。

供选择的答案

A：①用户界面测试 ②输入/输出测试 ③集成测试 ④单元测试

B：①需求分析 ②概要设计 ③详细设计 ④结构设计

C：①集成测试 ②可靠性测试 ③系统性能测试 ④强度测试

D：①编程 ②概要设计 ③维护 ④详细设计

E：①过程测试 ②函数测试 ③确认测试 ④逻辑路径测试

【解析】

一个软件在交付使用前，主要经历 3 种测试：单元测试、集成测试和验收测试。

单元测试也称为模块测试，是针对各个程序单元或模块单独进行的测试，通常称之为"分调"。单元测试在实验阶段进行，一般和程序编写结合起来，由程序员分工进行，并且多模块可以并行展开，单元测试着重发现和解决程序编写中的差错，比较重视对程序结构的检验，单元测试是整个测试阶段的基础。

集成测试把自己通过单元测试的模块连接起来，通常使用功能测试法，着重检验模块间的接口以及设计中的问题。集成测试所依据的模块说明书和测试方案应在概要设计阶段完成。集成测试的目的是发现设计错误。

确认测试也称验收测试，其目的是为了确认已开发的软件能否满足验收标准。确认测试是对软件质量的全面考核，验收测试以前应制定验收标准和验收测试工作。确认测试应在模拟的环境中在强度测试的基础上进行，测试计划应在软件需求分析阶段完成。

【答案】A：④ B：③ C：① D：② E：③

试题 18（1991 年试题 1）

从供选择的答案中选出应填入下列叙述中的{ }内的正确答案，把编号写在答卷的对应栏内。
软件测试的目的是 A。为了提高测试的效率，应该 B。使用白盒测试方法时，确定测试数据应根据 C 和指定的覆盖标准。一般说来与设计测试数据无关的文档是 D。软件的集成测试工作最好由 E 承担，以提高集成测试的效果。

供选择的答案

A：①评价软件的质量 ②发现软件的错误
③找出软件中的所有错误 ④证明软件是正确的

B：①随机地选取测试数据 ②取一切可能的输入数据作为测试数据
③在完成编码以后制定软件的测试计划 ④选择发现错误的可能性大的数据作为测试数据

C：①程序的内部逻辑 ②程序的复杂程度
③使用说明书 ④程序的功能

D：①需求规格说明书 ②设计说明书
③源程序 ④项目开发计划

E：①该软件的设计人员 ②该软件开发组的负责人
③该软件的编程人员 ④不属于该软件开发组的软件设计人员

【解析】

软件测试的工作量约占软件开发期的一半，是软件开发过程的一个重要阶段，直接影响软件的质量，软件测试的目的是为了发现软件的错误，提高测试效率，因此应该选择容易发现错误的数量作为测试数据。

测试数据的方法有两种：黑盒方法和白盒方法。黑盒方法将程序看作不能打开的黑盒，根据程序的功能或程序的外部特性测试数据。白盒测试方法是设计测试用例的一类方法，使用白盒测试方法，允许测试者检查程序的内部结构，以程序的内部逻辑为依据，设计测试数据和指定覆盖标准，因此白盒测试方法又称为逻辑覆盖方法。

一般来说，与设计测试数据有关的文档是需求规格说明书、设计说明书和源程序等文档，与项目开发计划无关。软件集成测试工作最好由不属于该软件开发组的设计人员承担。这样可以使参加测试的人员避免认为自己开发的程序总是正确这种"先入为主"的意识和"当局者迷"的现象，能够用更加客观的态度投入软件集成测试工作，提高测试的效果。

【答案】A：② B：④ C：① D：④ E：④

试题 19（1991 年试题 5）

从供选择的答案中选出应填入下列叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

软件设计中划分程序模块通常遵循的原则是要使各模块间的耦合性尽可能 A。3 种可能的模块耦合是：

B.例如，一个模块直接引用另一模块中的数据。

C.例如，一个模块把开关量作为参数传送给另一模块。

D.例如,一个模块把一个数值量做为参数传送给另一模块。其中 E 的耦合性最强。

供选择的答案

A：①强 ②适中 ③弱

B～E：①公共耦合 ②数据耦合 ③逻辑耦合
④外部耦合 ⑤内容耦合 ⑥控制耦合

【解析】

模块间的联系是评价程序结构质量的重要标准,模块间的互相关联程度由模块的耦合性衡量。耦合性越弱，模块间的联系越小，每个模块越容易独立地被理解、编写和修改；同时某个模块的错误不容易扩散蔓到其他模块，因此，在软件设计中，使各个模块的耦合尽可能弱将成为划分程序模块的原则。

模块按照其耦合性由强到弱可以分成以下 5 种：

(1)内容耦合 内容耦合指一个模块直接引用另一模块的内部信息；

(2)公共耦合 公共耦合指两个模块引用人同的全局数据区；

(3)控制耦合 控制耦合指一个模块传送给另一模块的参数是用于控制模块内部逻辑的控制信号；

(4)复合耦合 复合耦合指一个模块传送给另一模块的参数是一个复合的数据结构；

(5)数据耦合 数据耦合指一个模块传送给另一模块的参数是单个的数据项。

上述的 5 种耦合中内容的耦合性最强。

【答案】A：③ B：⑤ C：⑥ D：② E：⑤

试题 20 (1990 年试题 1)

从供选择的答案中选出应填入下列叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

软件可移植性是用来衡量软件的 A 的重要尺度之一。为了提高软件的可移植性，应注意提高软件的 B。采用 C 有助于提高 B。为了提高可移植性，还应 D。使用 E 语言开发的系统软件具有较好的可移植性。

供选择的答案

A：①通用性 ②效率 ③质量 ④人-机界面

B：①使用的方便性 ②简洁性 ③可靠性 ④设备独立性

C：①优化算法 ②专用设备 ③表格驱动方式 ④树型文件目录

D：①有完备的文档资料 ②选择好的宿主计算机
③减少输入/输出次数 ④选择好的操作系统

E：①Coblo ②APL ③C ④PL/1

【解析】

可移植性是衡量软件质量高低的一个重要尺度之一。可移植性是指把原先在某硬件或软件环境下正常运行的软件，移植以另一个硬件或软件环境下，使该软件也能正确运行的难易程度。

设备独立性是指软件对设备的访问不随设备特性的变化而变化，所以，提高软件的设备独立性是提高软件可移植性的重要手段。为提高软件的可移植性，在设计和编写程序时，应注意使程序代码不要随意地与具体环境直接关联，并且不要直接使用硬件环境中的设备，使程序代码尽可能与环境无关，否则，设备变了，程序也要做大量的改动。

对一类常规或有一定通用功能的设备来说，它们都有一套反映其特性的参数。对软件来说 设备之间的差异就是它们的参数不同。为使程序能使用不同参数的设备，可采用表格驱动。表格驱动是指程序内部保留有一张记录着设备特性参数的参数表，这样，软件移植或环境有所改变时，只需修改参数表中若干参数，在程序不作任何修改的情况下，软件便可在新的环境下正确使用。表格驱动方式使程序独立于设备，促进了程序可移植性的实现。

完备的文档资料有助于软件移植的顺利进行。软件移植时，对原来的软件有所修改是难免的，理解原来软件的设计思想和程序采用的算法都依赖于软件的文档资料。

与其创建高级语言相比，C 语言开发的软件具有良好的可移植性，计算机系统大多数都配置有 C 语言编译程序和提供 C 语言的标准库函数。统计资料表明，不同系统上的 C 语言编译程序 80%的代码是公共的。

【答案】A：③ B：④ C：③ D：① E：③

试题 21 (1990 年试题 5)

从供选择的答案中选出应填入下列叙述中的{ }内的正确答案，把编号写在答卷的对应栏内。

模块内聚度用于衡量模块内部各成分之间彼此结合的紧密程度。

1.一组语句在程序的多处出现，为了节省内存空间把这些语句放在一个模块中，该模块的内聚度是 A 的。

2.将几个逻辑上相似的成分放在一个模块中，该模块的内聚度是 B 的。

3.模块中所有成分引用共同的数据，该模块的内聚度是 C 的。

4.模块内的某成分的输出是另一些成分的输入，该模块的内聚度是 D 的。

5.模块中所有成分结合起来完成一项任务，该模块的内聚度是 E 的。它具有简明外部界面，由它构成的软件易于理解、测试和维护。

供选择的答案

A～E：①功能性 ②顺序性 ③通信性 ④过程性 ⑤偶然性 ⑥瞬时性 ⑦逻辑性。

【解析】

模块内聚度用于衡量模块内部各成分之间彼此结合的紧密程度。在软件设计过程中，设计者在划分程序模块之时，应力求模块的内聚度尽可能强。按模块内聚性分，模块内聚度由弱到强的顺序是：偶然性、逻辑性、瞬时性、过程性、通信性、顺序性和功能性。

一个模块内部的各组语句这宰没有必然的联系，则这个模块是偶然性内聚的；几个逻辑上相似的成分放在一个模块中，这种模块是逻辑内聚的；一些经常需要一起执行的程序段放在一起构成的模块称作瞬时性内聚的;因各成分要引用共同的数据，而把它们组成一个模块，这种模块称作是通信性内聚的；如模块中的一个成分的输出是另一些成分的输入，称这种模块是顺序性内聚的；如模块中的所有成分都是为了完成一项共同的任务,这种模块称作简明的外部界面,其可理解性、可测试性和维护性都比较好，所以功能内聚模块是设计人员应该追求的目标。

【答案】A：⑤ B：⑦ C：③ D：② E：①

2.5 数据库系统基础知识

2.5.1 主要知识点

掌握数据库模型、数据库系统结构和关系数据库结构的基础知识，熟练掌握 SQL 语言的使用 关系代数的运算。

2.5.1.1 关系数据库的数据体系结构

关系模型遵循数据库的 3 级体系结构：

(1)关系模式。数据库的概念模式定义为关系模式的集合。每个关系模式就是记录类型。关系模式的定义包括模式名、属性名、值域名和模式的关键。关系模式仅仅是对数据本身特性的描述。

(2)关系子模式是用户所用到的那部分数据的描述。除了指出用户用到的数据外，还应指出数据与模式中相应数据的联系，即指出子模式与模式之间的对应性。

(3)关系存储是作为文件看待的，每个元组就是一个记录。由于关系模式有键，因此存储一个关系可用散列方法或索引方法实现。

2.5.1.2 关系模型和关系运算

用二维表格结构表示实体集，关键码表示实体间联系的数据模型称为关系模型。在关系中，能唯一标识元组的属性集称为关系候选键，被选用的候选键称为关系的主键。关系中每一个属性对应一个取值范围，称为属性的值域。关系可以定义为元数(属性个数)相同的元组的组合。关系是一个集合，集合中的成分是元组，这些元组的属性个数应相同。

关系数据库的数据更新操作必须遵循实体完整性规则、引用完整性规则和用户定义的完整性规则。

关系查询语言根据其理论基础的不同分成两大类：一类是关系代数语言，其特征是查询操作是以集合操作为基础的运算；另一类是关系演算语言，其特征是查询操作是以谓词演算为基础的运算。

关系代数是以集合代数为基础发展起来的、以关系为运算对象的一组高级运算的集合。把关系看成集合，集合代数中的运算可以引入到关系运算中来，还有一些运算是针对关系数据库环境专门设计的。

关系代数的 5 种基本操作：并、差、笛卡儿积、投影和选择。

关系代数的 4 种组合操作：交、联接、自然联接和除法。

扩充的关系代数操作：外联接(左外联接和右外联接)、外部并(Out union)和半联接。

2.5.1.3 关系数据库 SQL 语言

结构化查询语言 SQL 是集 DDL、DML 和数据控制功能于一体的数据库语言。SQL 的 DML 是介于关系代数和关系演算之间的一种语言。

SQL 语言的组成：

• 一个 SQL 数据库是表的汇集，它用一个或多个 SQL 模式定义。

• 一个 SQL 表由行集构成，一行是列的序列，每列对应一个数据项。

• 一个表或者是一个基本表，或者是一个视图。基本表是实际存储在数据库中的表，而视图是由若干基本表或其他视图构成的表的定义。

• 一个基本表可以跨一个或多个存储文件，一个存储文件也可存放一个或多个基本表。

• 用户可以用 SQL 语句对视图和基本表进行查询等操作。

• SQL 用户可以是应用程序，也可以是终端用户。SQL 语句可嵌入在宿主语言的程序中使用，宿主语言可以是常用的高级语言。SQL 用户也能作为独立的用户接口，使交互环境下的终端用户使用。

SQL 包括了所有对数据库的操作，主要有数据定义(SQL DDL)、数据操纵(SQL DML)、访问控制和嵌入式 SQL。

2.5.1.4 关系数据库规范化理论

关系数据库规范化理论主要包括 3 方面的内容：

(1)函数依赖。指数据之间存在的各种联系和约束，例如建就是一种依赖。函数依赖是最基本的一种依赖。

(2)范式。模式分解的标准形式。关系模式分解的两个特性实际上涉及到两个数据库模式的等价性问题，包括数据等价和依赖等价两个方面。数据等价是指两个数据库实例应表示同样的信息内容，用"无损联接"衡量。依赖等价是指两个数据库模式应有相互逻辑关系的函数依赖集，此时数据的语义是不会出差错的。

(3)模式设计方法。设计规范的数据库模式的方法。
衡量关系模式的优劣的标准就是模式的范式(NF)。范式有许多种，与数据依赖有着直接的联系。

①第一范式(INF)

如果关系模式 R 的每个关系 r 的属性值都是不可分的原子值，那么称 R 是第一范式(INF)的模式
，r 是规范化的关系。LNF 的模式的关系数据库最基本的要求。

②第二范式(2NF)

我们把关系模式 R 的属性分为两类：一类是键的属性，称为主属性；另一类是不属于任何键的属性，称为非主属性。若关系模式 R 是 INF，且每个非主属性完全函数依赖于候选键，那么称 R 是 2NF 模式。

③第三范式(3NF)

若关系模式 R 是 INF，且每个非主属性都不传递依赖于 R 的候选集，那么称 R 是 3NF 模式。在 3NF 模式中排除了非主属性的对键的传递依赖。3NF 的模式必定是 2NF 的模式。局部依赖和传递依赖是产生冗余和异常的两个重要原因。由于 3NF 模式中不存在非主属性对候选键的局部依赖和传递依赖，因此具有较好的性能。而对于非 3NF 的 INF 和 2NF，甚至非 INF 的关系模式，于它们性能上的弱点，一般不宜作为数据库模式，通常需要将它们变换成 3NF 或更高级的范式
，这种变换过程，称为"关系的规范化处理"。

④BC 范式(简称 BCNF)
若关系模式 R 是 INF，且每个属性都不传递依赖于 R 的候选键，那么称 R 是 BCNF 模式。此时排除了任何属性对键的传递依赖。

上述 4 种范式之间关系：BCNF$\subset$ 3NF$\subset$ 2NF$\subset$ 1NF。

2.5.2 试题解析

高级程序员级考试中，数据库基础试题所占的比重一般，基本上每年一道题。从历年试题统计(见表 2-8)来看，主要考查与关系数据库有关的知识、关系模式、关系模式的范式、关系代数运算等是反复考查的内容，总的来说，难度一般都不大。

试题 1 (2000 年试题 7)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

域表达式{ab|R(ab)∧ R(ba)}转换为等价的关系代数表达式，所列出的式子中 A 是正确的.SQL 中集合成员资格的比较操作"元组 NOT IN(集合)"中的"NOT IN"与 B 操作符等价。SQL 中涉及属性 AGE 是否是空值的比较操作，写法 C 是错误的。类似于工资在 800 至 5000 之间"这种约束，是属于 DBS 的 D 功能。

设关系模式 Q 是 3NF 模式，那么，E 这种提法是不正确的。

供选择的答案

A: ①Л1,2(б$_{1=4\wedge 2=3}$(R×R)) ②Л1,2(б1=4($R\triangleright\!\triangleleft_{2-3}R$))

③ Л1,2(б1=4($R\triangleright\!\triangleleft_{2-1}R$)) ④ R∩Л1,2(R)

B: ①<>SOME ②=SOME ③<>ALL ④=ALL

C：①AGE IS NULL ②NOT(AGE IS NULL)
③AGE=NULL ④AGE IS NOT NULL

D：①完整性 ②并发控制 ③安全性 ④恢复

E：①Q 一定是 2NF 模式 ②Q 可能不是 4NF 模式
③Q 可能不是 BCNF ④Q 一定不是 BCNF

【解析】

问题 A 涉及域表达式与关系代数表达式的转换。域表达式{ab|R(ab) ∧ R(ab)}表示取出二元关系 R 中有对称关系的二元组的集合，即(a，b)∈R，(b，a) ∈R，如果用 D 来表示该域表达式，则有 D$\subseteq$R。答案中只有②是正确的。

$R\underset{2=3}{\triangleright\!\triangleleft}R$是关系 R 与其自身的条件连接，条件是第 1 个 R 的第 2 个元素与第 2 个 R 的第 1 个元素相等。举例说明如下：

R={(1，2)，(2，3)，(3，2)，(2，1)(3，4)}

则$R\underset{2=3}{\triangleright\!\triangleleft}R$ 的结果是

{(1，2，2，3)，
(1，2，2，1)
(2，1，1，2)
(2，3，3，2)
(3，2，2，3)
(2，3，3，4)}；

再作 δ 选择运算找出 1，4 列相等的元组{(1，2，2，1)，(2，1，1，2)，(2，3，3，2)，(3，2，2，3)}；

再作 π 投影运算取 1，2 列，得到{(1，2)，(2，1)，(2，3)，(3，2)}。

问题 B 涉及集合成员资格比较。SQL 中的集合比较有 4 种：集合成员资格比较、集合成员算术比较、空关系测试和重复元组的测试。其中集合成员资格比较有两种形式：

(集合 1)IN(集合 2)

(集合 1)NOT IN(集合 2)

这里"IN"与算术比较中的"=SOME"等介，"NOT IN"与算术比较中的"＜＞ALL"等价。

问题 C 涉及空值的处理。SQL 中允许属性值为空值，用关键字 NULL 表示空值。测试某属性值为空值，用"(某属性)IS NULL"表示，测试某属性值为非空值，用"(某属性)IS NOT NNULL 或"NOT((某属性)IS NULL)"来表示。

问题 D 涉及属性值的约束，属于数据库完整性范畴。

问题 E 涉及关系模式的范式。根据 4 种范式之间的关系：BCNF⊂3NF⊂2NF⊂1NF，可以判断 3NF 模式必定是 2NF 模式，BCNF 必定是 3NF 模式，但是 3NF 可能是 BCNF 模式，也可能不是 BCNF 模式。

【答案】A：② B：③ C：③ D：④

试题 2 (1999 年试题 7)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

最常用的一种基本数据模型是关系数据模型，它用统一的 A 结构来表示实体及实体之间的联系。关系数据库的数据操作语文(DML)主要包括 B 两类操作。

关系运算以关系代数为理论基础，关系代数的最基本操作是并、差、笛卡尔积和 C。用 R⋈S 表示关系 R 和关系 S 的 D 。

设关系 R 和关系 S 图示如下：

R：

A	B	C
a	b	c
b	b	f
c	a	d
d	a	d

S：

B	C	D
b	c	d
b	c	e
a	d	b

T：

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
d	a	d	b

则关系 T 是关系 R 和关系 S E 的结果。

供选择的答案

A：①树 ②网络 ③图 ④二维表

B：①插入和删除 ②检索和更新 ③查询和编辑 ④统计和修改

C：①投影、联接 ②联接、选择 ③选择、投影 ④交、选择

D：①联接 ②笛卡尔积 ③ 联接 ④自然联接

E：①自然联接 ② 联接 ③笛卡尔积 ④并

【解析】

用二维表格结构表示实体类型、关键码表示实体间联系的数据模型称为关系模型。关系数据 库的 DMLA 语言分成查询语句和非查询语句，前者描述用户要进行的各种检索操作，后者描用户要进行的有关数据库更新的操作。

关系运算以关系代数为理论基础，关系代数的最基本操作是并、差、笛卡儿积、选择和投影。设有两个关系 R 和 S，则 R 和 S 的并是由属于 R 或属于 S 的元组组成的集合，记为 RUS。R 和 S 的交是由既属于 R 又属于 S 的元组组成的集合，记为 R∩S。R 和 S 的差是由属于 R 但不属于 S 的元组组成的集合，记为 R-S。设关系 R 和 S 的元数分别为 r、s，则 R 和 S 的笛卡儿积是一个(r+s)元的元组集合，每个元组的前 r 个分量来自 R 的一个元组，后 s 个分量来自 S 的一个元组，记为 R×S。投影操作是对关系进行垂直分割，消去关系中某些列，并重新按排列的次序，再删除重复的元组。选择操作是根据某些条件对关系做水平分割，选择符合条件的元组。

关系 R 和 S 的自然联接用 R⋈S 来表示，其计算过程如下：

设 R 和 S 的公共属性为 A1……Ak，挑选 R×S 中满足 R·A1=S·A1…R·Ak=S·Ak 的元组；去掉 SA1…SAk 列。自然联接是构造新关系的有效方法，是关系代数中常用的一种运算。从试题中 E 的计算结果看，这里应该是经过自然联接得到的结果。

【答案】A：④ B：② C：③ D：④ E：①

试题 3 (1988 年试题 2)

从供选择的答案中选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

实体联系模型(简称 ER 模型)中的基本语义单位是实体和联系。ER 模型的图形表示称为 ER 图。联系可以同 A 实体有关。实体与实体之间的联系可以是 B 。

利用 ER 模型进行数据库的概念设计，可以分成 3 步：首先设计局部 ER，然后把各个局部 ER 模型综合成一个全局的模型，最后对全局 ER 模型进行 C ，得到最终的 ER 模型。

ER 模型向关系模式的转换规则是把一个实体类型转换成一个关系模式，实体的属性是关系的属性，实体的键是关系的键。把一个联系类型转换成一个 D ，参与该联系类型的各实体的键以及联系的属性转换成 E ，其中的键由实体与实体之间的联系决定。

供选择的答案

A：①0 个 ②1 个或多个 ③1 个 ④多个

B：①一对一和一对多 ②一对一和多对多
③一对多和多对多 ④一对一、一对多和多对多

C：①简化 ②结构化 ③最小化 ④优化

D：①联系模式 ②数据模式 ③关系模式 ④逻辑模式

E：①联系属性 ②关系的属性 ③数据属性 ④关系的候选键

【解析】
实体联系模型中，联系指的是实体之间的联系，实体之间的联系有一对一、一对多和多对多 3 种。如：一个公司有一个经理，而每个经理只有一个公司任职，则公司与经理之间是一对一联系；一个公司有多个副经理，而每个副经理只在一个公司任职，则公司与副经理之间是一对多联系；一个老师可以有多个学生，而一个学生也可以有多个老师，则老师与学生之间是多对多的联系。联系可以同一个或多个实体有关。

利用 ER 模型进行数据库的概念设计，分为 3 步：第 1 步设计局部的 ER 模型；第 2 步综合各个局部 ER 模型成为全局的 ER 模型；第 3 步对全局的 ER 模型进行优化，得到最终的 ER 模型。

ER 模型向关系模式的转换依据如下：把一个实体类型转移为一个关系模式，相应实体的属性转换为对应的关系的属性，实体的键是关系的键；把一个联系类型转换为一个关系模式，所有与该联系相关的实体的键及联系的属性转换成关系的属性，关系模式的键由实体与实体之间的联系所决定。

【答案】A：② B：④ C：④ D：③ E：②

试题 4 (1997 年试题 5)

从以下叙述中选出 5 条最确切的叙述，把相应编号依次写在答卷的 A～E 栏内。

①在数据库系统中，数据独立性指数据之间的相互独立，互不依赖。

②SQL 语言的视图定义和视图操作功能不支持逻辑数据的独立性。

③SQL 语言中不提供显式地使用索引的功能，支持了物理数据的独立性。

④用户对"脏数据"的读出是由于数据库完整性规则受到了破坏。

⑤在数据库系统中，数据的安全性是指保护数据以防止未被授权用户的蓄意或者无意使用。

⑥实体完整性规则指主关键字值的任何组成部分都不可以是空值;引用完整性规则则不允许引用不存在的实体(即元组)

⑦在数据库系统中，数据的完整性是指数据的正确性和有效性。

⑧"授权"是数据库系统中采用的完整性措施之一。

⑨事务处理(Transaction)是数据库运行的基本单位。如果一个事物处理成功，则全部数据得到更新和提交；如果失败，则已做的全部更新被恢复成原状，好像整个事务处理示进行过一样。这样使数据库保持了一致性。

⑩对数据库的查找、增添、删除和修改等操作都需由数据库管理员进行完整性定义和安全性授权，由数据库系统具体执行。

【解析】

该题主要查考生对数据库基本概念的掌握情况,下面从基本概念入手进行逐条解析。

(1)数据的独立性分为物理独立性和逻辑独立性。物理独立性指当数据库的物理布局和物理组织形式改变时，不影响数据库的全局逻辑结构的性质；逻辑独立性是指当数据库的全局逻辑结构改变时，不影响某些局部逻辑结构的性质，因而题中①不正确。视图是数据库中满足一定条件约束的数据组成的虚拟关系，视图可作为某用户的专用数据部分，这样既提高了数据的独立性，又利于数据的安全保密，所以题中②认为视图定义产不支持逻辑数据的独立性是不正确的，而叙述③是正确的。

(2)数据的完整性是指保证数据的正确性和有效性，因而叙述⑦是正确的。可以采取多种方法来保证数据的完整性。实体完整性规则指主关键字值的任何组成部分都不可以是空值；引用完整性规则则不允许引用不存在的实体(即元组)，叙述⑥是正确的。

(3)并发控制是为了防止由于多个用户并行地对数据操作时，他们之间会相互干扰从而导致数据库数据不一致，为此需对并发操作采取控制措施。用户对"脏数据"的读出是因为并发控制没做好，所以叙述处理④不正确。事务处理(Transaction)是数据库运行的基本单位，是数据库系统中保证数据一致必的手段。如果一个事务处理成功，则全部数据得到更新和提交；如果失败，则已做的全部更新被恢复成原状，好像整个事务处理未进行过一样。这样使数据库保持了一致性。叙述⑨正确。

(4)在数据库系统中，数据的安全性是指保护数据以防止未被授权用户的蓄意或者无意使用。"授权"是数据库进行安全保护的措施之一(即允许某一用户以某种方式访问某些数据)，而不是完整性措施，所以叙述⑧不正确，叙述⑤正确。

(5)对数据库的查找、增添、删除、修改等操作都需由数据库管理员进行完整性定义和安全性授权，由数据库系统具体执行。除数据库管理员以外，设计、建立和管理维护数据库的软件人员可参与数据库系统的分析和设计，在数据库运行期间对用户的使用、存取进行监控并统计数据库使用情况，在必要时整理并重新构造数据库或恢复数据库。叙述⑩不够准确。

【答案】A：③ B：⑤ C：⑥ D：⑦ E：⑨

试题 5 (1996 年试题 3)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

设有一图书管理数据库，其关系模式是 R0(L#，B#，BNAME，BPRICE，BPUB)，其属性分别表示个人借书证号、书号、书名、书价、图书出版社。该关系模式 A 。它的主要问题是数据冗余。如把 R0 分解成两个关系模式 R1 B 和 R2 C ，则可以部分地解决这一问题。R1 和 R2 是规范化程度较差的范式 D 。另外一种分解方法可以得到 3 个模式 R3(L#，B#)，R4(B#，BNAME)、R5(BNAME，BPRICE，BPUB)，则 R3、R4 和 R5 都 E 。

供选择的答案

A、D、E：①属于第一范式但不属于第二范式 ②属于第二范式但不属于第三范式

③属于第三范式 ④不是范式
⑤属于第二范式但不属于第一范式 ⑥属于第三范式但不属于第二范式

B、C：①(L#，B#，BPRICE) ②(L#、B#)
③(B#，BNAME) ④(B#，BNAME，BPRICE，BPUB)
⑤(BNAME，BPRICE，BPUB) ⑥(L#，BNAME，BPRICE)

【解析】

本题与 1993 年试题 2 类似。

假定借书证号与书号是主码，可以唯一决定一条记录。由关系模式 R0 可分解成 3 个模式 R3、R4，R5，而得到第三范式，由书名可确定书价与图书出版社，而书号决定书名。

第一范式要求每个属性都是不可分解的，题中关系模式 R0 的所有的属性都不可再分，因而属于第一范式，但它不是第二范式，因为第二范式要求每个非主属性的完全函数依赖于主码，而 R0 中如果将(L#，B#)作为主码的话，则属性 BNAME 只是部分函数依赖于主码(因为它的完全函数依赖于 B#)，所以它不是第二范式。

如果有多人借同一种书，则 BNAME 等信息要重复出现多次。如果将 R0 分解成 R1(L#，B#)和 R2(B#，BNAME，BPRICE，BPUB)两个关系模式，则 R1 和 R2 都成为第二范式，但在 R2 中存在关系 B#→BNAME，BNAME/B#，BNAME→BPUB，因而非主属性 BPRICE B#)，R4(B#，BNAME)和 R5(BNAME，8PRICE，BPUB)，这样每一关系模式都是第三范式，在最大程度上降低了数据冗余。

【答案】A：① B：② C：④ D：② E：③

试题 6 (1995 试题 2)

从供选择的答案中，选出应填入下面叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

传统 的数据库基本上是由 A 组成的。B 在技术和理论上已经成熟，成为当前商用数据库的主流。C 技术是 80 年低中期引入的。目前，多媒体数据库基本上靠与关系模式相结合的 D 来支持。但当数据量大，数据结构复杂时，靠 D 很难适应。当前，在 DBMS 的研究方面，较活跃的是 E 。

供选择的答案：

A、D：①图形 ②文件 ③元组 ④文件系统
⑤对象 ⑥过程

B：①关系数据库 ②网状数据库 ③层次数据库 ④空间数据库

C：①关系数据库 ②网状数据库 ③层次数据库 ④面向对象数据库

E：①网状数据库 ②层次数据库 ③DBASE ④演绎数据库

【解析】

数据是描述事物的符号记录。数据库是存放数据的仓库，是长期储存在计算机内、有组织的、可共享的数据集合。传统的数据基本上是由大量记录所组成，记录又称为元组。

在数据库中是用数据模型对现实世界进行抽象的，现有的数据库系统均是基于某种数据模型的，数据模型可以对数据进行抽象表示，是我们进行数据模型化的工具。数据库领域中最常用的数据模型有 3 种，它们是层次模型、网状模型和关系模型。其中层次模型和网状模型统称为非关系模型。

数据库按数据模型来分，可分为层次型数据库、网状数据库和关系数据库 3 大类，层次和网状数据库又称为非关系数据库。这 3 类数据库都是在 60 年代末发展起来的。关系数据库由于采用二维表的形式来描述实体和实体间的联系，其数据结构相对简单，对于用户来讲浅显易懂，技术理论上也已经成熟，已成为当前常用数据库的主流。

随着数据库应用领域的不断发展、扩充，传统的数据库已不能完全适应新领域应用的需要。这些领域包括计算机辅助设计(CAD)，计算机辅助基础(CAM)、VISI设计、计算机辅助软件工程(CASE)、图像处理和多媒体技术等方面。除了对传统数据库的需求外，这些领域还要求数据库能够处理更复杂的结构、递归定义的对象和更大的数据量等，具有更多更复杂的数据库随着这种需求而出现，空间数据库、面向对象数据库、多媒体数据库和演绎数据库等就是其中发展起来的一部分。

空间数据库是随着地理信息系统发展起来的一种数据库。空间数据库的空间数据是用来表示空间物体的位置、形态、大小以及分布特征等信息的数据。空间数据不仅包含物体本身的空间位置和位置信息，还包含物体空间即拓扑关系的信息。空间数据库常用的数据结构有矢量和栅格结构两种。空间数据库研究的主要内容有空间数据表示、数据结构与数据模型空间、数据库管理系统等。空间数据库的研究正在深入系统地进行，但在理论与技术上还不如关系数据库那么成熟。空间数据库的应用领域主要用于地理信息系统，它是地理信息系统的核心。

面向对象数据库是从 80 年代中期开始出现并发展的，它是关系数据库系统和面向对象的方法相结合组织起来的，这种新一代的数据库系统称为面向对象数据库系统(ODBC)。面向对象数据库是符合面向对象数据模型的，其数据结构是建立在对象和类的概念上的，数据具有封装性、继承性和多态性等特点。当前面向对象数据库系统的研究工作主要集虽在面向对象数据模型的形式化与面向对象的程序语言的结合等方面。

多媒体数据系统是一种由文本、图像、声频和和视频等多种形式的数据组织起来的集成数据库系统。目前，相当一部分多媒体数据库是通过关系模式与文件系统相结合来支持的，但当数据量大、数据结构复杂时，靠文件系统是很难适应的。分布式多媒体数据库系统能对对象进行存储、检索以及支持对象间的通信。这些对象是由图像、文本、声频和视频等数据类型混合而成的。多媒体数据库的多媒体数据模型、系统实现等问题目前正在深入研究。多媒体数据库的发展也与存储介质的发展、输入输出介质的发展、数据模型理论的发展、通信介质的发展等密切相关。多媒体数据库系统改善了用户的工作环境，提高了用户对数据的可利用性。

演绎数据库的数据库模型是一种基于逻辑的模型。Horn 子句的逻辑为演绎数据库提供了递归定义的能力，从而可以定义更复杂的数据，支持更强的数据操作能力以及提供更完善的完整性保护，并提供数据操作与宿主语言统一的说明性语言。因此，演绎数据库具有比传统数据库更强的能力。演绎数据库是数据库技术与逻辑程序和人工智能相结合的结果。目前，相对于传统数据库的研究而言，演绎数据库的研究更为活跃。

【答案】A：③ B：① C：④ D：④ E：④

试题 7 (1995 试题 16)

从下列与数据库中数据的独立性、完整性和安全性有关的叙述中，选出 5 条正确叙述，并按编号从小到大的次序写在答卷的 A～E 栏内。

①在数据库系统中，数据独立性指的是数据之间相互独立，互不依赖。

②数据库系统中，由于有封锁机制，所以应用程序对数据的存储结构和存取方法有较高的独立性。

③SQL 语言的视图定义和视图操作功能在一定程度上支持了逻辑数据独立性。

④SQL 语言中不显式提供索引功能，这是对物理数据独立性的支持。

⑤在数据库系统中，数据的完整性是指数据的正确性和相容性。

⑥"授权"是数据库系统中采用的完整性措施之一。

⑦实体完整性和参照完整性是可应用于所有关系数据库的两条完整性准则。

⑧"脏"数据的读出是数据库安全性遭到破坏的一个例子。

⑨在数据库系统中，数据的安全性是指保护数据以防止不合法的使用。

⑩SQL 语言在 COMMIT 语句、ROLLBACK 语句和 LOCK TABLE 语句都具有维护数据库安全的功能。

【解析】

数据库系统中，数据独立性是一个重要概念。数据的物理存储结构和数据在计算机内表示的逻辑结构千差万别，都有多种不同形式。如果用户不必考虑数据在计算机中的表示及其在计算机中的物理存储结构，而是按自己所需的逻辑结构对数据进行操作，就大大地方便了用户；另一方面，若数据在计算机内的逻辑结构是独立于物理存储结构的，即计算机对数据进行操作时，物理结构的变化不会影响数据整体的逻辑结构，这就大大提高了机器的效率。

上述两个方面体现了两层独立性的概念：逻辑数据的独立性(即用户的数据逻辑结构独立于计算机内数据表示的逻辑结构)，物理数据的独立性(即计算机机内的数据表示的逻辑结构独立于物理存储结构)。为了实现这两层数据独立性，把数据库的结构分为外部级(单个用户视图级)、概念级(全局视图)和内部级(存储视图)三级。用户通过数据操作语言进行的操作是对外部级的操作；而外部级数据模式与概念级数据模式的转换、概念级数据模式与内部级数据模式的转换均由数据库管理系统实现。显然，SQL 语言的视图定义和操作功能在一定程度上支持了逻辑数据的独立性；SQL 不显式提供索引同是对物理数据独立性的支持。

本题③、④两条叙述是正确的，第①条叙述是错误的。数据的独立性并不是指数据之间互不依赖，事实上许多数据之间是有依赖关系的。第②条叙述也是错误的，因为封锁机制是解决并发操作问题的，而不是解决数据的结构问题的。

数据库系统中，数据的完整性是指数据的正确性和相容性，数据库管理系统提供完整性保护的功能，系统提供定义完整性约束条件的功能和检查完整性约束条件的方法。系统的完整性子系统就是根据完整性约束条件工作的。完整性约束条件包括对数据值和结构的约束，还包括对数据在操作前后应满足的约束等。实体完整性和参照完整性可适用于关系数据库。

因此本题中⑤、⑦两条叙述是正确的，而第⑥条是错误的。因为"授权"是授于用户对数据库中何种数据文件作何种操作的权力，而为了防止非法使用数据库，是数据库系统中采用的安全性措施，而不是完整性措施。数据库系统中，数据的安全性是指保护数据以防止不合法使用，避免数据泄露或遭到破坏。它通常是通过对用户标识、鉴别及对存取进行控制来实现的，"授权"就是保证安全性的一种措施，因此，本题第⑨条叙述是正确的，第⑧条叙述是错误的。因为"脏"数据一般是指由于并发控制不当，由操作异常所形成的数据，而不是不合法操作所形成的数据。第⑩条也是错的。因为 SQL 语言 LOCK TABLE 是用于并发控制的，而不是用于安全性控制的。

[答案]A：③ B：④ C：⑤ D：⑦ E：⑨

试题 8（1994 年试题 1）

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

在数据库理论中，关系 R 和 S 在第 i A 和 B 上的 θ-联结(Jion)写成 $R\underset{i\theta}{|\times|}S$，其中 θ 是 C 。若 R 是 r 元关系，则有 $R\underset{i\theta}{|\times|}S$ =D，D 中的运算符×为 E 乘积符。

供选择的答案

A、B：①行 ②列 ③个记录 ④张表

C：①算术运算符，如+、- ②逻辑运算符，如 ∨、∧
③算术比较运算符，如=、＜ ④集合运算符，如 ∩、∪

D:① $\delta_{(I+j)\theta r}(R\times S)$ ② $\delta_{(I+j)\theta j}(R\times S)$ ③ $\delta_{i\theta(r+j)}(R\times S)$ ④ $\delta_{i\theta j}(R\times S)$

E：①算术 ②笛卡尔 ③矢量 ④逻辑

【解析】

关系代数是由一组以关系为运算对象的特定运算组成的，通过这组运算，对一个或多个关系进行分解和组合，构造出新的关系，从而得到所需要的数据。关系代数的运算可分为两类，一类是传统的集合运算，即并、差、交和笛卡尔积；另一类是特殊的集合运算，即投影、选择、联接、自然连接和除法。

关系代数是关系数据库的数学基础，是离散数学的重要组成部分，关系代数结合数据库，为关系数据库的全面开发奠定了基础，关系代数语言是关系型数据库的数据操作语言(DML)中重要的一类。

联接(Join)是关系代数运算中的一种，记号为 $R\underset{i\theta}{|\times|}S$ ，它是从两个关系 R 和 S 的笛卡尔积中选取属性间满足一定条件的元组。其中 i 和 j 分别是关系 R 的第 i 列分量和关系 S 中的第 j 列分量。 为算臬比较运算符，它包括″＜″、″＞″、″=″等运算符；当 为″=″时，上式称为等值联接。若有 $R\underset{i=j}{|\times|}S$ 即表示 R 与 S 的笛卡尔积中的第 r+j 列，实际上就是 S 中的第 j 列。

因此又有 $R\underset{i\theta}{|\times|}S=\delta_{i\ (r+j)}(R\times S)$，其中 δ 表示投影运算，×表示笛卡尔积。

【答案】A：② B：② C：③ D：③ E：②

试题 9 (1994 年试题 3)

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

数据库是存储在一起的相关数据的集合，能为各种用户所共享，且 A 。在关系数据库中，若关系模式中的每个关系的属性值均是不可分解的，则该关系模式属于 B 。
关系代数运算是以 C 为基础的运算，其 5 种基本运算是并、差、D 、投影和选择；规范理论研究中，分解 E 主要是消除 E 中多余的数据相关性。

供选择的答案

A：①消除了数据冗余 ②降低了数据冗余度
③具有不相容性 ④由用户控制物理数据的存取方式

B：①1NF ②2NF ③3NF ④BCNF

C：①代数运算 ②关系运算 ③谓词演算 ④集合操作

D：①交 ②连接 ③笛卡尔积 ④自然连接

E：①内模式 ②视图 ③外模式 ④关系模式

【解析】

数据库是存储在一起的相关数据的集合，它能为各种用户所共享，并具有最小冗余度，数据间联系密切而又有较高的数据与程序的独立性。关系数据库是以关系型数据模型为基础的数据库，关系模型就是通过表格结构来表示实体类型及实体间联系的模型。

关系模型中将表格结构中的记录类型为关系模式，将表格中的字段称为属性，字段值称为属性值，将记录称为元组。关系模型中，一个数据库模式是一个关系模式的集合。对同一问题，可以选用不同关系模式集合作为数据库模式，但其性能的优劣大不相同，为了区分优劣，把数据

库模式分为各种不同等级的范式。

第一范式(1NF)：关系模式中每个关系的属性值均不再可分。

第二范式(2NF)：若某个关系是第一范式，且每个非主属性完全函数依赖于各关键字。

第三范式(3NF)：若某个关系是第二范式，且每个非主属性不传递依赖于任何关键字。

BC 范式（BCNF）：若某个关系是第一范式，且每个属性均不传递依赖于任何关键字。

可知这些范式之间的关系为：BCNF>3NE>C:2NE>C>1NE.

在数据库设计中，分解的关系模式使其性能优化，实质上，分解关系模式主要是消除关系模式中多余的数据相关性，降低数据的冗余度是设计的主要目标之一，但实际上不可能完全消除数据的冗余，否则会出现操作上的问题。

关系数据库的操作语言分成关系代数语言主导关系演算语言两类，关系代数语言中的运算是以集合为基础的运算，把关系看成元组的集合来处理，其基本运算有并、差、笛卡尔积、投影和选择五种。

[答案]A：② B：① C：④ D：③ E：④

试题 10 （1993 年试题 1）

从下列有关数据库的叙述中，选出 5 条正确的叙述，并把编号按从小到大的次序写在答卷的对应栏内。

① 关系代数的最基本操作有并、差、笛卡尔乘积、选择和投影。

② 视图是用户看到的数据库。它由一个或多个基本表导出。其定义存在于数据库目寻中；其数据在物理上以表的形式直接存储。因此对视图就像对基本表一样能进行查、添、删、改等操作。

③ 一般的完整性规则有两个：实体完整性规则是指主关键字值的任何组成部分都不是空值；引用完整性规则是如果关系的 R 的属性 A 为外关键字（设为关系 S 的主关键字），则 A 的每个值是 S 的主关键字的某一值或是空值。

④ 对于查、添、删、改操作都需由数据库管理员 DBA 进行完整性定义和安全性授权，由数据库系统具体执行。

⑤ 多用户的数据库系统的目标之一使它的每个用户好像面对着一个单用户的数据库一样使用它，为此数据库管理系统必须进行并发控制。

⑥ 数据库系统目录（或称数据库字典）也由一些关系组成，所以用户同样可以对其进行查、添、删、改操作。

⑦ 在 SQL 的查询语句中，要对所查询的数据指明存取路径，进行导航，数据库管理系统依此执行代数优化和非代数优化，这样才能有效地访问数据。

⑧ 嵌入式的数据库语言构成的应用程序环境包括主语言（如程序设计 C 和 Fortran 和数据子语言（如 SQL），前者能处理记录和域，后者只能处理表，游标机制起着两种语言的桥梁作用。

⑨ 事务(Transaction)是数据库运行的基本工作单位。如果一个事务执行成功，则全部更新提交；如果一个事务执行失败，则已做过的全部列新被恢复原状，好像整个事务从未有过这些更新。这样就保持了数据库处于一致性状态。

⑩ 既然数据库能实现对不同用户数据共享，所以数据库中数据不应该存在任何冗余。

[解析]

①正确。并、差、笛卡尔积、选择和投影这 5 种运算是基本的关系代数运算，是关系代数运算的最小完备集。

②错误。视图是一种虚拟关系，由数据库中的某一个或几个基本关系通过关系运算导出。视图燕不是以实际的数据库，在数据库中只存储该视图的定义，即从有关的基关系导出该视图的方法。

当用户对数据库中的数据进行存取时，数据库管理系统自动地将其转换为对相应的基关系的存取，然后再执行变换的命令。在关系数据库管理系统中，提供用户使用数据库的不同要求，允许不同的用户从不同的角度看同一个数据库，这是能过视图来实现的。

③正确。在关系数据模型中，一般有实体完整性规则和关联(引用)完整性规则。实体完整性规则是指在任何关系的任何一个元组中，主关键字值的任一分量都不允许为空值。关联完整性规则是指如果某一关系 R 中的一个属性 A 相对于另一关系 S 为外关键字，则 A 的值必须要么是空值，要么等于 S 中某一个关键字值。

④错误。数据完整性定义不一定需要由数据管理员进行。

⑤正确。数据库技术的最大优点之一是数据的共享性，但同时需解决并发控制问题。比如当某一个应用程序正在对某个记录进行修改的过程中，另一程序恰好要

读取该记录，，这时读得的数据就可能是不正确的，为了避免这种情况的发生，就必须对并发操作施加某些控制措施，如记录加锁。

⑥错误。系统的全部工作领带于数据字典中数据的正确性，因此不能允许用户随意定义和操纵数据字典中的表。

⑦错误。SQL 是一种过程性语言。对于非过程性语言，由系统来分析该语句应该干什么，然后选择最佳方案，实现该语句。

⑧正确。当查询语句产生多元组结果关系时，为了能把该组元组逐个地提交给宿主语语句处理，数据库管理系统一般都提供了游标概念，包括游标关系和游标指针。

游标机制把结果关系看作一种特殊视图，称为游标关系，用游标指针指明当前所处理元组的位置，通过用游标操作语句改变游标指针，从而逐个地从游标关系中取出元组，赋给宿主语言程序变量，进行信息交流。

⑨正确。为便于维护数据的完整性，数据库管理系统把需要进行多步数据操作才能完成的一项业务称作事务。数据库管理系统规定，事务是数据库中独立执行的最小程序单位。一个事务中的各个数据操作不是独立存在的，它们共同组成一个有机整体，或者全部执行，或者全部不执行。如果某个事务的某一中间步骤发生了错误，则数据库管理系统将自动恢复数据库原始状态，以保证数据的完整性。

⑩错误。在数据库中必要的数据冗余是必须的。

[答案]A：① B：③ C：⑤ D：⑧E：⑨

试题 11 （1993 年试题 2）

从供选择的答案中，选出应填入下面关于关系数据库叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

设有关系模式 W（C，P，S，G，T，R），其中各属性的含义是：C--课程，P--教师，S--学生，G--成绩，T--时间，R--教室，根据语义有如下数据依赖集：

D={C→P，（S，C）→G，（T，R）→C，（T，P）→R，（T，S）→R}

关系模式 W 的一个码（关键字）是 A，W 的规范化程度最高达到 B。若将关系模式 W 分解为 3 个关系模式 W1（C，P）、W2（S，C，G）、W3（S，T，R，C）。则 W1 的规范化程度最高达到 C，W2 的规范化程序最高达到 D，W3 的规范化程序最高达到 E。

供选择答案

A：①（S，C）②（T，R）③（T，P）④（T，S）

B ～E:①1NE ②2NE ③3NE ④BCNE ⑤4NE

[解析]

设 R（W）是一个关系模式，X，Y⊆W。关系模式 R 上的函数依赖是形式 f:X→Y 的一个命题。其含义是为对于 R 的任意一个可能的实例 r,如果对任意 t2,t2r，t2[x]=t2[x]，则必有 t1=[y]=t2[y].X⊆W 是 R 的一个关键字，如果存在函数依赖 X→W，且不存在 X⊆W，使得 X→W 也成立。

由以上定义可知，（T，S）是 W（C，P，S，G，T，R）的一个关键字。如果 R 中的每一个属性的值域中的每一个值都是不可分解的,则称 R 属于第一范式(1NE)：如果 R 是第一范式,并且 R 中任何一个非主属性都完全函数依赖于 R 的每一个候选关键字，则称 R 是第二范式（2NE）；如果 R 是第一范式，并且 R 中不存在任何非主属性传递函数依赖于 R 的某个候选关键字，则称 R 是第三范式（3NE）；如果 R 是第一范式,并且 R 中不存在任何属性传递函数依赖于 R 的任何一个关键字，则 R 属于 Boyce-Codd 范式（BCNF）；如果对于在 R 上成立的每一个非平凡多值函数依赖 X→→Y，X 都是 R 的一个超关键字，则 R 属于第四范式（4NE）。

由以上定义可知,W1 的规范化程度最高达到 4NE,W2 的规范化程序最高达到 4NE,W3 的规范化程序最高达到 2NE 。

[答案]A：④ B：② C：⑤ D：⑤ E：②

试题 12 （1991 年试题 4）

从供选择的答案中选出应填入下列叙述中□的正确答案,把编号写在答案的对应栏内。

一个数据库系统必须能表示实体和关系。关系可与 A 实体有关。实体与实体之的关系有一对一、一对多对多三种，其中 B 不能描述多对多的联系。

一般地，一个数据库系统 C 外视图 D 概念视图，E 数据子语言。

供选择的答案

A：①0 个 ②1 个 ③2 个 ④3 个或 3 个以上 ⑤1 个或 1 个以上 ⑥0 个或 0 个以上

B：①网状模型 ②层次模型 ③关系模型 ④网状模型和层次模型 ⑤层次模型和关系模型 ⑥网状模型和关系模型 ⑦网状模型、层次模型和关系模型

C～E①只能有一种 ②最多只能有两种 ③至少有两种 ④可以有多种

[解析]

实体是一个范围广泛的概念，从具体的人、物、事件到抽象的状态以及概念，都可以用实体来抽象表示。实体是存在于现实世界中，并且可以根据其自身信息加鉴别的任何事物的抽象表示。联系是现实世界内部或者事物之间语义关系的抽象定义。联系可以同一个或者多个实体有关。实体之间的联系可以是一对一（夫妻关系）、一对多（师生关系）或者多对多的（学生和课程关系）。实体和联系组成模型可以用 E-R 图表示。

层次模型、网状模型和关系模型是数据库的 3 种主要数据库模型。层次模型一种可以用树形表示的分层模型，它有两个特征：一是有且仅有一个无前驱的结点（即树根）；二是共他结点仅有一个前驱。所以，层次模型不能够描述多对多的联系。网状模型是有向图模型。关系模型以集合论为基础，关系是实体的元组集合。

关系数据库中，允许不同的用户从不同角度查看同一个数据库，这是视图来实现的。视图是一种中以用数据库内的一个或多个基关系，通过关系运算导出虚拟关系的方式。用户所看到的是外视图，外视图并不实际地存储在数据库内。外视图所反映的数据库都要通过与概念视的映射，变换到概念数据中。因此，一个数据库可以提供多个不同的外视图，但只存在一个概念视图同物理数据库一一对应。一个数据库系统，由于它的每个模式都有其独立的数据子语言，所以一个数据库可以支持多个数据子语言。

[答案]A：⑤ B：② C：④ D：① E：④

试题 13 （1990 年试题 3）

从下列关于数据库系统特点的叙述中，选出 5 条正确的叙述，把编号依次写在答卷对应栏内。

① 数据库避免了一切数据重复。

② 数据库减少了数据冗余。

③ 各类用户程序均可随意地使用数据库中的各种数据。

④ 用户程序按所对应的子模式使用数据库中的数据。

⑤ 数据库数据可经 DBA 认可的各用户所共享。

⑥ 数据库系统中如概念模式有所改变，则需将与其有关的子模式做相应改变，否则用户程序需改写。

⑦ 数据库系统中概念模式如有改变，子模式不必变，因而用户程序亦不必改写。

⑧ 数据库系统的存储模式如有改变，则概念模式应予调整，否则用户程序会在执行中出错。

⑨ 数据库系统的存储模式如有所改变，概念模式无需改动。

⑩ 数据一致性是指数据库中数据类型一致。

[解析]

①错误。数据库中最在限度地减少数据冗余，但不能避免一切数据重复，有时为了提高效率还会增加重复数据。

②正确。数据库能减少数据冗余。

③错误。数据库系统的安全性措施使每个用户的数据得到合理的保护，防止任何未经许可的其他用户对数据的访问和修改。

④正确。数据库的逻辑结构可分为子模式、概念模式和存储模式三级，其中子模式是用户看到和使用的数据描述，是用户与数据库的接口，常称作用户对数据库的视图。用户使用 DML 对数据库进行操作，实际是对子模式记录进行操作，用户程序按子模式使用数据库。概念模式是数据库中全部数据整体逻辑结构描述。存储模式是数据库中的物理存储结构的描述.存储模式是数据库的物理存储结构的描述。概念模和存储模式对用户是透明的。

⑤正确。数据共享是数据的目的之一，但为了数据的安全性，用户共享数据必须得到 DBA（即数据库管理员）的授权。

⑥错误。在数据库的三级模式结构中，概念模式的改变不影响子模式，也不会影响以子模式方式使用数据库的用户程序，子模式相对于概念模式是独立的。概念模式的改变需改变子模式与概念模式之间的映射关系。

⑦正确。理由同⑥。

⑧错误。如同⑥所述，概念模式相对于存储模式也是独立的，存储模式的改变也不会影响概念模式，而改变的是概念模式与存储模式之间的映射关系。

⑨正确。数据库系统的存储模式如有改变，概念模式无需改动。

⑩错误。数据库中，数据一致性是指表示同一数据的多个副本之间没有矛盾，完全一致。

[答案]A：② B：④ C：⑤ D：⑦ E：⑨

2.6 多媒体基础知识

2．6．1 主要知识点

掌握多媒体的基本概念，熟悉图形、图像格式和基本音频、视频知识。

2．6．1．1 常见的音频格式

（1） WAVE，扩展名为 WAV。

该格式记录了声音的波形，只要采样率高、采样字节长、计算机速度快，利用该格式记录的声间文件能够和原声基本一致。WAVE 的唯一缺点就是文件太大，毕竟它要把声音的每个细节都记录下来，而且不压缩。

（2） MOD，扩展名为 MOD，ST3，XT，S3M，FAR 和 669 等。

MOD 是一类音乐文件的总称，逐渐发展产生了 ST3，XT，S3M，FAR 和 669 等扩展格式，而其基本原理还是一样的。该格式的文件不仅存放了乐谱，而且存放了乐曲使用的各种音色样本，具有回放效果明确、音色种类永无止境的优点。

（3） MPEG-3，扩展名为 MP3。

MPEG-3 压缩较大，是一种有损压缩，其实际音质并不完美。在网络、可视电话能信方面，MP3 大有用武之地。由于本质不同，所以它没法和 MOD、MIDI 相提并论。从 HIFI 角度上讲，MP3 有损失，而 MOD 和 MIDI 则没有。

（4） Real Audio，扩展名 RA。

强大的压缩量和极小的失真度使其在众多格式中脱颖而出。与 MP3 相同，它也是为了解决网络传输带宽资源设计的，因此其主要目标是提高压缩比和容错性，其次才是音质。

（5）Creative Musical Format，的扩展名为 CMF。

这是 Creative 公司的专用音乐格式。它和 MIDI 差不多，只是音色、效果上有些特色，专用于 FM 声卡。不过其兼容性差，且效果无法和别的格式相提并论。

（6）CD Audio 音乐 CD，扩展名为 CDA。

CDA 格式就是唱片采用的格式，又叫"红皮书"格式，记录的是波形流。CDA 的缺点是无法编辑，文件长度太大。

（7）MIDI，扩展名为 MID。

作为音乐工业的数据通信标准，MIDI 可谓是一种非常专业的语言，它能指挥各种音乐设备的运转，而且具有统一的标准格式，甚至能够模仿用原始乐器的各种演奏持巧无法演奏的效果。MIDI 文件长度非常小。MIDI 的一个缺点是不能记录语音。

2．6．1．2 多媒体数据压缩的编码技术标准。

（1）H．261

H．261 是用于音频视频服务的视频编码和解码器（也称 PX64 标准）。应用目标是可视电话和视频会议系统。含有此标准的系统必须能实时地按标准进行编码和解码。H.261 与 JPEG 及 MPEG 标准间有明显的相似性,区别是 H.261 是为动态使用而设计的,并提供完全饮食的组织和高水平的交互控制.

(2)JPEG

JPEG 是静止图像压缩和解压缩算法的标准.JPEG 成为 ISO 的国际标准.

(3)MPEG

MPEG-X 版本是指一组 ITU 和 ISO 制定发布的视频、音频和数据的压缩标准。现在有 3 个版本的 MPEG：MPEG-1、MPEG-2 和 MPEG-4。MPEG 在以下 3 方面优于其他压缩/解压缩方案：具有很好的兼容性；MPEG 能够比其他算法提供更好的压缩比，最高达 200：1；更重要的是，MPEG 在提供高压缩比的同时，数据压缩的损失很小。

（4）DVI

DVI 视频图像压缩算法的性能与 MPEG1 相当，即图像质量可达到 VHS 的水平。压缩后的图像数据率约为 1。5Mbit/s.为了扩大 DVI 技术的应用，Intel 公司最近又推出了 DVI 算法的软件解码算法，称为 Indeo 技术。它能将数字视频文件压缩为五分之一到十分之一。Indeo 技术使用了多类"有损"和"无损"压缩技术。

2．6．2 试题解析

现在多媒体基础知识也列入新的大纲，1999 年度和 2000 年各有一道专门考查多媒体知识的试题，分别考查了音频知识和多媒体压缩技术，此前这方面的试题很少。复习时应掌握基本概念，熟悉有关的多媒体文件容量、量化方面的计算。

试题 1 （2000 年试题 8）

从供选择的答案中，选出应填入下面叙述中{ }内最确切的解答，把相应编号写在答卷的对应栏内。

数据压缩技术是多媒体信息处理中的关键技术之一，数据压缩技术可分为 A 两大类。 B 是一种与频度相关的压缩和编码方法 ，C 主要用于视频信息的压缩，

D 则常用于静止图片的信息压缩。由三基色（RGB）原理出发的 RGB 彩色空间，在多媒体技术中最常用的，此外还有多种彩色空间，但 E 不是计算机上用的彩色空间。

供选择的答案

A：①可逆与不可逆 ②高速与低速 ③编码与非编码 ④冗余与非冗余

B：①MIPS ②ISDN ③Huffman ④Gauss

C:①MIPS ②MPEG ③JPEG ④JIPS

D:①MIPS ②MPEG ③JPEG ④JIPS

E:①YUV ②HIS ③XYZ ④IMG

[解析]

本题主要考查多媒体数据压缩技术，还涉及到彩色空间表示。

一般将数据压缩技术分为无损压缩和有损压缩两大类。无损压缩和用数据的统计冗余进行压缩，可完全恢复到原始数据，没有任何失真，因此是可逆的。有损压缩利用人的视、听觉器官对图像或声音的某些频率成分不敏感的特性，在压缩过程中损失一定信息，不能完全恢复原始数据，因此是不可逆的。

Huffman 编码就是一种基于统计的无损压缩方案，是一种与频度有关的压缩编码方法。实际上的问题 B 提供的答案中，只有"Huffman"是压缩编码方法。

JPEG 是静止图像压缩和解压缩算法的国际标准。MPEG 一组由 ITU 和 ISO 制定发布的视频、音频、数据的压缩标准。

由三基色（RGB）原理出发的 RGB 彩色空间，在多媒体技术中是最常用的，此外还有多种彩色空间，如 YUV、HIS、CIEXYZ、CIELAB、CCIR601-2YcbCr。问题 E 提供的答案"IGB"不是彩色空间方面的术语。

[答案]

A: ① B:③ C: ② D:③ E:④

试题 2 (1999 年试题 8)

从供选择的答案中,选出相应填入下面叙述中{ }内的最确切的解答,把相应编号写在答卷的对应栏内.

在多媒体的音频处理中,由于人所敏感的声频最高为 A 赫兹(Hz),因此,数字音频文件中对音频的采样频率为 B 赫兹(Hz)。对一个双声道的立体声，保持 1 秒钟声音，其波形文件所需的字节数为 C ，这里假设每个采样点的量化数为 8 位。

MIDI 文件是最常用的数字音频文件之一，MIDI 是一种 D ，它是该领域国际上的一个 E 。

供选择的答案

A：①50 ②10k ③22k ④44k

B:①44.1k ②20.05k ③10k ④88k

C:①22050 ②88200 ③176400 ④44100

D:①语音数字接口 ②乐器数字接口 ③语音模拟接口 ④乐器模拟接口

E：①控制方式 ②管理规范 ③通信标准 ④ 输入格式

[解析]

本题考查多媒体音频的基本常识，非常简单。

在多媒体的音频处理中，采样频率是决定音频质量的一个重要因素。人的听觉带宽一般为 20Hz～20kHz,人敏感的声频最高为 22kHz。目前根据音频质量所确定的频率范围如下：

电话话音为 200Hz～3.4kHz

调幅广播为 50Hz～7kHz

调频广播为 20Hz～15kHz

宽带音频为 20Hz～20kHz

而常用的音频采样频率为 8kHz、2.025kHz、16kHz、22.05kHz、37.8kHz、44.1kHz,数字音频文件中对音频的采样频率为 44.1kHz.

信号编码的位数是决定音频质量的另一个重要因素,它决定数字采样的可用动态范围和信噪比。

对一个双声道的立体声，由于两个声道，在每个采样点的量化位数为 8 位的情况下,保持一秒钟的声音码,则包含 $2\times44.1\times10^3$ 个采样点的数据,那么就是 88200 个字节。

MIDI 的全名为 Musical Instrument Digital Interface,即乐器数字的接口，泛指数字音乐的国际标准。MIDI 的标准规定了电子乐器与计算机连接的电缆和硬件，还指定了在装置间传送数据的通信协议。任何电子乐器，只要有处理 MIDI 信息的处理器和适当的硬件接口，都能够成为 MIDI 装置。利有 MIDI 文件演奏音乐，所需存储量很小，演奏 2 分钟乐曲的 MIDI 文件，文件大小不到 8kB.。

[答案]A：③ B：① C：② D：② E：③

试题 3 (1996 年试题 10)

从供选择的答案中,选出应填入下面叙述中{ }内最确切的解答,把相应编号写在答卷的对应栏内.
多媒体技术的关键在于解决动态图像和声音的存储与传输问题.若不经压缩,以 VGA640×480 点阵存储一幅 256 色的彩色图像大约需 A MB 存储空间,以 9600bit/s 的速度传输这幅图像大约需 B 秒,按我国电视 PAL 标准每秒 25 幅,一张 650MB 的光盘可容纳约 C 秒的这样图像画面,播放时传送速率应不低于每秒 D MB.模拟声音数字化存放是通过采样和量化实现的,若采样频率 44.1kHz,每个样本 16 位,存放一分钟双声道的声音约占 E MB 存储空间.

供选择答案

A～E: ①0.3 ②1.4 ③2.4 ④7.5 ⑤10
⑥32 ⑦78.6 ⑧87 ⑨98.4 ⑩256

[解析]

因为 640×480 点阵存储一幅 256 色彩色图像所需的存储空间为 $256=2^8$,所以存储 256 色后个点的信息需 1Byet 存储空间,一幅图像的存储量 =640×480×1Byte=307200Byte≈0.3MB,问题 A 的答案应选①.

传输一幅图像的时间为 307200Byte/(9600bit/s)=256s,(注:bit/s 为比特/秒,1 字节为 8bit),所以问题 B 的答案应选⑩.

650MB 光盘可容纳(650MB/0.3MB)/(25 副/秒)=2167 副/（25 副/秒）=87 秒的图像。问题 C 应选⑧.
播放时传送速率为 0.3MB/幅×25 幅/秒=7.5MB/秒,所以问题 D 的答案应为④.

已知声间的采样率为 44.1kHz,样本 16 位,双声道,则存放一分钟声音的存储空间为 44.1kHz×166bit×2×60 秒=84.7Mbit/8=10.6MB,所以问题 E 的答案为⑤.

[答案]A:① B:⑩ C:⑧ D:④ E:⑤

试题 4 (1995 年试题 24)

从供选择的答案中,选出相应填入下面关于 CAD 中彩色产生方法叙述中{}内的正确答案,把编号写在答卷的对应栏内.

彩色图形显示和印刷设备采用三原色方法.CRT 彩色显示器选用 RGB(红、绿、蓝)三原色，是 A 混色系统。彩色喷墨等彩色图形印刷设备选用 CMY（青、品红、黄）三原色，是 B 混色系统，黑色在这两种系统中的表示是 C 。

某种颜色的补色是从白色减去该颜色成分后所呈现的色彩。CMY 分别是 RGB 的补色。CRT 显示的纯红色，在用彩色喷墨印刷输出时其 CMY= D ，而白纸上的青色图形在 CRT 显示器中其 RGB= E 。

供选择的答案：

A、B：①减色 ②加色 ③比例乘积色 ④互补色 ⑤荧光 ⑥油墨 ⑦光电发射⑧印刷

C：①RGB=000，CMY=000 ②RGB=000，CMY=111
③RGB=111，CMY=000 ④RGB=111，CMY=111

D、E：①000 ②001 ③010 ④011 ⑤100 ⑥101 ⑦110 ⑧111

[解析]

不同颜色通过配色产生新的颜色。适当选择 3 种不同颜色相加，可以配成几乎所有可见的颜色，这就是三原色原理。三原色的构成是红、绿、蓝。

在 CRT 彩色显示器中，通过靠得很近的 RGB（红、绿、蓝）3 个发光点产生混色，就是相加混色系统。当 RGB=111 时便呈现白色。人们看到印在纸上的颜色，这是纸反射部分的入射光呈现的颜色。当入射光包含全部可见光颜色（白色），而所用纸吸收了红色光，它反射的便是红色的补色即青色（C）了。这也可看作反射光是由绿和蓝两种颜色相加组成，这便是青色颜料的作用。同样，品红色（M）颜料和黄色（Y）颜料，分别是吸收了白色入射光中绿不色和蓝色后呈现的颜色，这是减色混色系统。当同时使用 3 种颜色 CMY=111 时，它吸收了白色入射光中的红、绿、蓝三色，自然呈现黑色。

RGB 加色混色系统工程的颜色若为 CMY 减色混色系统表示时，可用公式（1）计算。同样 CMY 减色混色系统中的颜色若用 RGB 加色混色系统表示时，可用公式(2)计算。

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} R_{白} \\ G_{白} \\ B_{白} \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \ldots\ldots(1)$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} C_{黑} \\ M_{黑} \\ Y_{黑} \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \quad \ldots\ldots(2)$$

其中白色在 RGB 中是 R 白=G 白=B 白=1，黑色在 CMY 中是 C 黑=M 黑=Y 黑=1.例如 RGB 中的纯红色是 RGB=100,在 CMY 中便是 CMY=011.白色光被吸收了绿色和蓝色,反射光呈红色.同样在 CMY 中的青色是 CMY=100,在 RGB 中便是 RGB=011,由绿色和蓝色相加得青色.

[答案]A:② B:① C:② D:④ E:④

试题 5 （1993 年试题 9）

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

多媒体技术是当前计算机发展的一个热门方向。这里，多媒体的含义主要是指 A 等多种 B 。它强调多媒体信息的 C 。C 是多媒体发展中要解决的关键技术之一。在 SRAM、CD-ROM、磁带和高密度软盘 4 种存储器中，当前最适合用来存储多媒体信息的是 E。

供选择的答案

A：①如磁带、磁盘、光盘 ②如双绞线、同轴电缆、光纤
③传输信息的介质 ④表达信息的形式

B:①输入输出的设备样 ②存储信息的实体 ③传输信息的介质 ④表达信息的形式

C：①分时处理 ②批处理 ③综合与集成处理 ④分布式处理

D：①压缩技术 ②可靠性技术 ③流水线技术 ④加密技术

E：①SRAM ②CD-ROM ③磁带 ④高密度软盘

[解析]

多媒体信息的综合与集成处理，能将文字、声音、静态图形等有机地结合起来。如可将图像经过处理后嵌入图形，配上文字再与声音合成一体。用多媒体计算机模拟施行时，足不出户就能看到动态景物，阅读到景点的文字说明，领略到音响的效果。

声音和活动的图像都是模拟信息，要想将其在计算机中存储与处理，需要对信息进行数字化处理。压缩技术是多媒体发展中要解决的关键技术，压缩信息量，需要有大容量的存储器。在本题中给出的 SRAM、CD-ROM、磁带和高密度软盘 4 种存储器中，只有 CD（Compact Disk）-ROM（Read Only Memory ）,即只读光盘最合适用来存储多媒体信息。CD-ROM 的容量可达数百澌兆字节，与压缩技术相结合可存储相当长时间的动态图像，很有实用价值。

[答案]A：③ B：④ C：③ D：① E：②

2. 硬件基础知识试题精解

第三章 硬件基础知识试题精解

与程序员及试题相比，高级程序员级硬件基础知识试题份是量还有小一些，试题难度也不大。存储器系统知识是考查的重点内容，I/O 系统、虚拟内存、Cache 检错校验码、系统可靠性等知识点值得关注。

1990-2000 年硬件基础知识试题,按涉及知识点统计,考查体系结构和主要部件的有 12 道题,考查存储器系统的有 9 道题,考查安全性与可靠性的有 6 道题,考查体系结构其他基础知识的有 5 道题,还有 3 道综合性试题涉及及并行处理、系统性能评价以及操作系统、网络。

3.1 计算机体系结构的主要部件

3．1．1 主要知识点

掌握机内代码、算术运算和逻辑运算等计算机基础知识，重点掌握海明校验码、循环冗余校验码，系统结构，CPU，I/O 系统和总线结构等方面的知识。由于涉及的知识点较多，这里主要介绍 I/O 系统，其他内容就不一一介绍了。

3．1．1．1 输入输出控制器

输入输出控制器，也称 I/O 模块，负责控制外部设备与主存储器、CPU 的寄存器之间的数据交换。它通过系统接口（内部接口）与主机（CPU、主存储器）交互，通过设备接口（外部接口）与各种外设打交道。

其主要功能有：①控制与定时。I/O 系统必须对通信进行控制和定时，以协调外设和内部资源之间的通信流量，因为内部资源如主存、系统总线等被一系列的操作包括数据的输入输出所共享。②与 CPU 通信。③与设备通信。I/O 系统与外设之间交换控制命令、状态住处以及数据。④数据通信缓冲。I/O 系统需要数据缓冲寄存器来暂存数据，以此来适应通信双方速率的不匹配，从而提高主机利用率。⑤错误检测机制。外设和主机进行数据交换的过程中可能会产生错误，通常使用奇偶校验码、CRC 循环冗余校验码进行检测和纠正。

3．1．1．2 输入输出的工作方式。

输入输出系统主要有 3 种方式与主机交换数据。

（1） 程序控制方式

输入输出完全由 CPU 控制,在整个 I/O 过程中 CPU 必须等待而不能进行其他工作，无法充分发挥 CPU 高速的处理能力。CPU 发出 I/O 命令，命令中包含了外设的地

址信息和所要执行的操作，相应的 I/O 系统将执行命令并设置状态寄存器。CPU 不停地定期地查询 I/O 系统以确定该操作是否已经守成。当有多个外设需要和主机交换数据时，CPU 必须定期轮询外设，以确定每个外设的状态。其查询方式可以分为两类：串行点名和并行查询。

（2） 程序中断方式

CPU 利用中断方式完成数据的输入/输出。当 I/O 系统与外设交换数据时，CPU 无需等待也不必查询 I/O 的状态，而可以处理其他任务。当 I/O 系统完成了数据传输后，则以中断信号通知 CPU。CPU 保存正在扭亏为盈行程序的现场后，转入 I/O 中断服务程序完成与 I/O 系统的数据交换，再返回原主程序继续执行。与程序控制方式相比，中断方式因为 CPU 无需等待而提高的效率。
在系统中具有多个中断源的情况下，常用处理方法有多中断信号线法、中断软件查询法、雏菊链法、总线仲裁法和中断向量表法。

（3） DMA 方式

CPU 只是在数据传输前和完成后才介入,而数据的传输过程由 DMA 控制器来管理，无需 CPU 参与。数据直接写入或读出主存储器，不再经 CPU 中转。

中断法虽然比程序控制法更加有效，但两者都是由软件来完在数据的传输，难以胜任高速的数据传输需求。在主存和 I/O 模块之间进行数据交换仍需由 CPU 控制，并且每一数据都需经过 CPU 中转。因此不仅影响了 CPU 的工作，也降低了 I/O 数据传输率。

DMA 方法使用 DMA 控制器（DMAC）来控制和管理数据传输。DMAC 和 CPU 共享受系统总线，并且具有独立访问存储器的能力。在进行 DMA 传输时，CPU 放弃对系统总线的控制而由 DMAC 来控制总线，由 DMAC 提供存储器地址及必需的读写控制信号，实现外设与存储器之间的数据交换。

DMAC 获取总线的 3 种方式：

①暂停方式。DMAC 请求 CPU 让出系统总线，由 DMAC 使用，直到一组数据全部传送完毕后，再把总线控制权还给 CPU。

②周期窃取方式。请求 CPU 进入空闲态，暂时放弃总线，插入一个 DMAC 周期，传送完一个字后，把总线还给 CPU。

③共享方式。在 CPU 不使用总线时，DMAC 进行 DMA 传输。

DMA 的 3 种组织方式：

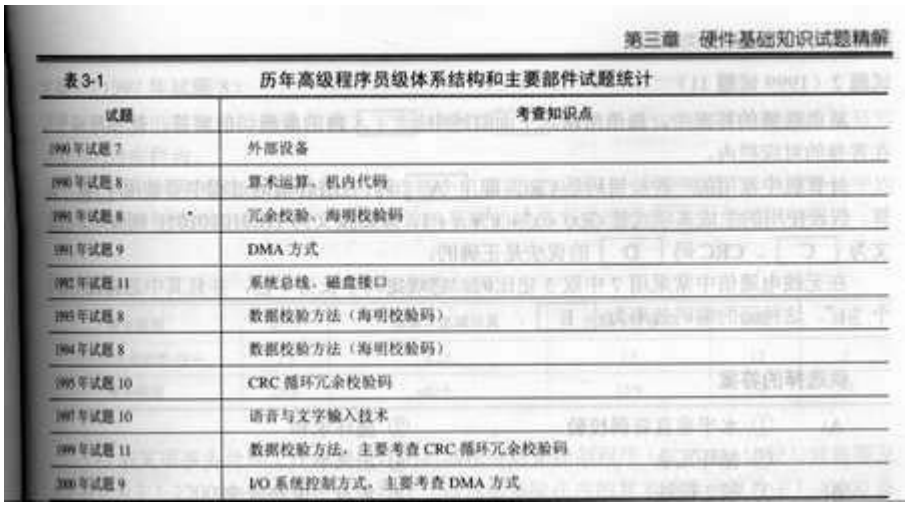
① 单总线分离 DMA 方式；

② 单总线集成 DMA 方式；

③ I/O 总线方式

3.1.2 试题解析

根据考试大纲，存储器系统知识已作为独立的一个大项，所以有关计算机体系结构和主要部件的内容相对少了，归到此类的试题也就不多了。从历年试题统计（见表 3-1）来看，数据校验方法（特别是海明码和循环冗余校验码）、I/O 系统控制方式（特别是 DMA）是反复考查的重点，而考查具体外部设备知识的试题很少，这也反映了对高级程序员的要求较高。值得一提的是，2000 年度试题 9 与 1991 年试题 9 考查的内容基本相同。



第三章 硬件基础知识试题精解

表 3-1 历年高级程序员级体系结构和主要部件试题统计

试题	考查知识点
1990 年试题 7	外部设备
1990 年试题 8	算术运算、机内代码
1991 年试题 8	冗余校验、海明校验码
1991 年试题 9	DMA 方式
1992 年试题 11	系统总线、磁盘接口
1993 年试题 8	数据校验方法（海明校验码）
1994 年试题 8	数据校验方法（海明校验码）
1995 年试题 10	CRC 循环冗余校验码
1997 年试题 10	语音与文字输入技术
1999 年试题 11	数据校验方法，主要考查 CRC 循环冗余校验码
2000 年试题 9	I/O 系统控制方式，主要考查 DMA 方式

试题 1 （2000 年试题 9）

从供选择的答案中，选出应填入叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

直接存储器访问（DMA）是一种快速传送大量数据常用的技术。工作过程大致如下：

1. 向 CPU 申请 DMA 传送。

2. 获 CPU 申请 DMA 控制器接管 A 的控制权。

3. 在 DMA 控制器的控制下，在存储器和 B 之间直接进行数据传送，在传送过中不需要 C 的参与。开始时需提供要供送的数据的 D 和 E 。

4. 传送结束后，向 CPU 返回 DMA 操作完成信号。

供选择的答案

A：①系统控制台 ②系统总线 ③I/O 控制器 ④中央处理器

B：①外部设备 ②运算器 ③缓存 ④中央处理器

C：①外部设备 ②系统时钟 ③系统总线 ④中央处理器

D：①结束地址 ②起始地址 ③设备类型 ④数据速率

E：①结束地址 ②设备类型 ③数据长度 ④数据速率

[解析]

这是一道考查 DMA 方式的概念题，内容单一，解答起来容易，具体知识请参见 3。1。1。2 节有关 DMA 的内容，以及本节试题 10（1991 年试题 9）。

[答案]A：② B：① C：④ D：② E：③

试题 2 （1999 试题 11）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

计算机中常用一种检错码是 CRC，即 A 码。在进行编码过程中要使用 B 运算。假设使用的生成多项式是 G（x）=x^4+x^3+x+1。原始报文为 11001010101，则编码后的报文为 C 。CRC 码 D 的说法是正确的。

在无线电通信中采用 7 中取 3 定比码，它规定码字长为 7 位，并且其中总有且仅有 3 个″1″。
这种码的编码效率为 E 。

供选择的答案

A：①水平垂直奇偶校检 ②循环求和 ③循环冗余 ④正比率

B：①模 2 除法 ②定点二进制除法 ③二-十进制除法 ④循环移位法

C：①1100101010111 ②110010101010011 ③11001010101011100
④110010101010101

D：①可纠正一位差错 ②可检测所有偶数位错 ③可检测所有小于校验位长度的突发错④可检测所有等于、小于校验位长度的突发错

E：①3/7 ②4/7 ③$\log_2 3/\log_2 7$ ④（$\log_2 35$）/7

[解析]

计算机在存储和传送数据的过程中，为了保证数据的准确性，一般要进行数据校验和纠错。CRC（循环冗余）码是一种常见的校错码。

假设被校验的二进制信息表达 C（X）表示，则 C（X）=CK-2…C1C0 取值为 0 或者 1。将 C（X）左移 n-k 位，则可表示成 $c(x)*2^{n-k}$，这样 C（X）的左边就会空出 n-k 位，这 n-k 位就是校验码的位置。

CRC 码就是用 $c(x)*2^{n-k}$ 除以生成多项式 g(x)，所得到的余数即为所求的校验位。

在本题中，生成多项式是 110011，按照上面的算法进行运算，就能够得到余数 0011，将其添加到原始报文的末尾，就得到编码后的报文。

无线电中常采用的 7 中取 3 定比码，规定码字长为 7 位，并且其中总有且仅有 3 个"1"，那么它的编码效率是这样计算的：

首先，7 位中有且仅有 3 个"1"的数据可以有 C^3_7 个，即（7*6*5）/（3*2*1）=35；

其次，取以 2 为底的对数，得到 $\log_2 35$;

最后，除以总位数 7，得到$(\log_2 35)/7$。

这样我们就可以看出求编码效率的公式：（$\log_2$(码字数)）/总位数。

[答案]A：③ B:① C: ② D:④ E:④

试题 3 (1997 年试题 8)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

某 CPU 的主振频率为 100MHz,平均每个机器周期包含 4 个共振周期.各类指令的平均机器周期数和使用频度见表 3-2,则该计算机系统的速度为平均约 A 兆指令/秒.

表 3-2 1997 年试题 8

指令类别	访内	一般算术逻辑运算	比较与转移等	乘除	其他
平均机器周期数/指令	2.5	1.25	1.5	15	5
使用频度	25%	40%	25%	5%	5%

(1) 若某项事务处理工作所要执行的机器指令数是控制程序(以访内比较与转移等及其他指令为主)220000 条指令和业务程序（以包括乘除在内的算术逻辑运算为主）90000 条指令，且指令使用频度表 3-2，则该计算机系统的事务处理能力约为 B 项/秒。

(2) 若其他条件不变，仅提高主振频率至 150MHz，则此时该计算机速度平均约为 C 兆指令/秒，对上述事务的处理能力约为 D 项/秒。

(3) 若主频仍为 100MHz，但由于采用了流水线和专用硬件等措施，使各类指令的每条指令平均机器周期数都变为 1。25，此时，计算机的速度平均约 E 兆指令/秒。

供选择的答案：
A～E：①1 ②5 ③10 ④15
⑤33.3 ⑥50 ⑦66.7 ⑧100

[解析]

指令周期(Instruction Cycle)指的是执行一条指令所需的时间；总线周期(Bus Cycle,也称作主振周期)指的是 CPU 从存储器或 I/O 端口存取一个字节所需的时间；时钟周期（Clock Cycle，也称作主振周期）指的是 CPU 处理动作的最小单位。指令周期划分一个个总线周期，不同的指令所需的总线周期也不相同，有的只需一个，有的只需要若干个，一个总线周期通常包括 4 个时期周期。

CPU 的主振频率(时钟频率)为 100MHz,即该 CPU 每秒有 100M/4=25M 个主振周期.该计算机的平均指令周期为 2.5*25%+1.25*40%+1.5*25%+15*5%+5*5%=2.5 个主振周期,所以该计算机的速度平均约为 25M/2.5=10M 指令/秒.所以问题 A 的答案为③.

某项事务处理工作所要执行的控制程序的平均指令周期为:

(2.5*25%+1.5*25%+5*5%)/(25%+25%25%)=2.2727

业务程序的平均指令周期为:

(15*5%+1.25*40%)/(5%+40%)=2.7777

完成一项事务处理工作所需的机器周为:

220000*2.2727+90000*2.7777=499994+249993=749987

这样该计算机系统的事务处理能力约为 25M/749987=33.33 项/每秒.所以问题 B 的答案为⑥.

若其他条件不变,仅提高计算机主频至 150MHz,此时该 CPU 每秒有 150M/4=37.5M 个主振周期,该计算机的速度平均约为 37.5M/2.5=1.5M 指令/秒.对上述事务的处理能力为 37.5M/749987=50.00 项/每秒.所以问题 C 的答案为④,D 的答案为⑦.

若主频仍为 100MHz,但采用的流水线及专用硬件等措施,使得各类指令的平均机器周期数都变为 1.25,则此时该计算机的速度平均约为 25M/1.25=20M 指令/秒.所以问题 E 的答案为⑤.

[答案]A:③ B:⑥ C:④ D:⑦ E:⑤

试题 4（1997 年试题 10）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

语音与文字输入技术是研究如何将计算机主要由键盘输入文字数据的方式逐步改变成由人们口述或写入的方式送入信息.语音与文字输入的过程包含有 A、预处理、特征抽取、B 与分类决策等环节。A 阶段通过传感器获得的模拟电信号要经过模数据转换变成数字信号，它需要对模拟电信号抽样测量，将测量的值 C ，以便用二进制数字信号来表示。预处理的主要任务是进行削弱无用住处和增强有用信息的工作。特征抽取将上述环节产生的输入样本以有利于决策的形式表示出来，典型的常用方法是 D 表示法。B 又称为分类器学习。分类决策则通过比较和决策来完成对输入信息的 E 。

供选择的答案

A～E：①辨认识别 ②特征向量 ③频谱分析 ④分级取整
⑤消除噪声 ⑥模型生成 ⑦数据合成 ⑧信息获取

[解析]

语音与文字输入技术是研究如何将人们口述或书写的信息直接输入计算机。实现其输入过程的系统构成如图 3-1 所示。

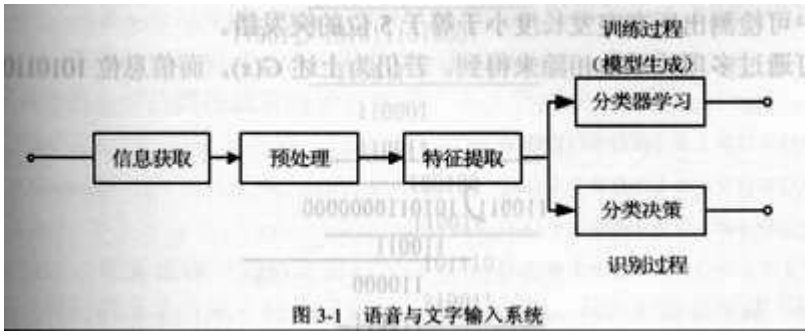


图 3-1 语音与文字输入系统

各部分的主要功能如下：

信息获取是将输入的语音与文字利用传感器转换成电信号，并从模拟量转换成数字量，它要对模拟电信号抽样测量，将测量的值分级取整。

预处理的主要作用是去除信号中的噪音，并增强有用信息。特征抽取（也称为特征提取）是指将经过预处理的信息输入计算机之后，从中提取能代表该信息特点的一组代码或向量，与在此之前已存入机内模型库的标准语音或文字的特征向量进行比较，从库中找出与输入向量最相似的标准语音或文字的特征向量进行比较，从库中找出与输入向量最相似的标准语音或文字作为最终的输入语音或文字，这一比较过程称之为匹配。完成了匹配过程也就完成了输入过程，因此如何提取特征向量是很重要的。

特征向量是根据文字或语音的特点按某一规律转换生成的。不同人写的文字或说话的口音都有所不同，因此增加了语音和文字输入技术的难度。从以上说明可以看到，在生成模型库时需要提取特征向量，作为语音或文字的样本；在应用时需要提取人的语音或文字或语音的特征几量，代机器识别用。分类器学习是指机器通过训练（学习），提取各个已知文字或语音的特征向量，建立模型库。分类决策是指对人输入信号进行辨认识别，最终完成语音或文字的识别工作。

[答案]A：⑧ B：⑥ C：④ D：② E：①

试题 5 （1995 年试题 10）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

某循环冗余码（CRC）的生成多项工为 G（X）=X^5+X^4+X+1，则它对任意长度的信息位产生 A 位长的冗余位，并可检验出所有突发长度 B 位的突发错误。若信息位于为 1010100，采用此生多项产生冗余位，加在信息位后形成的码字是 C 。在读出或接收端读到的码字中若不满足某种规律则可判断其必然有错。例如 D 和 E 就是出错的码字。

供选择的答案：

A、 B：①小于等于 4 ②4 ③小于等于 5 ④5 ⑤大于等于 5 ⑥大于 5

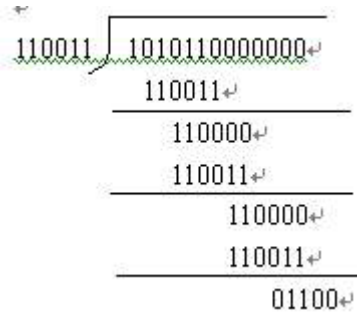
C～E：①1010110111111 ②1010111010001 ③1010110001101 ④1010111101010 ⑤1010111011001 ⑥1010110001100

[解析]

循环冗余码中的冗余位是由其生成多项式 G（X）去除信息多项工后得到的余式所决定的。若生成多项式为 K 次多项式，则余式为 K-1 次多项式，它对应的冗余位（即余式）的系数为 K 位。本题中 G（X）=X^5+X^4+X+1 为 5 次多项式，则冗余位为 5 位长。

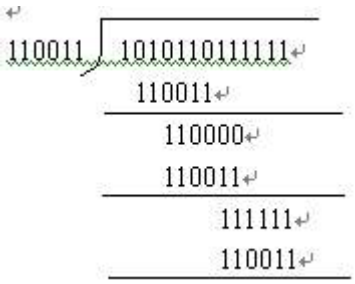
可以证明由K次生成多项式产生的冗余码能检测出所有突发长度小于等于K位的突发错误，故本题可检测出所有突发长度小于等于 5 位的突发错觉。

冗余地位可通过多项式系数相除来得到。若仍为上述 G(X)，而信息位 10101100，则除法过程如下：



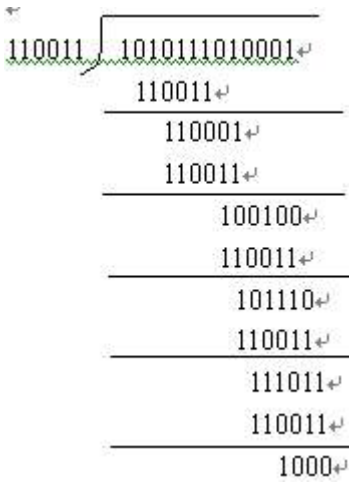
故冗余位为 01100，它加在信息位后的码字为 1010110001100。这里，需要指出 3 点值注意的地方。第一，上述除法中，被除数不是信息位，而应为信息位后面加上 K 位 0（这里 K=5）。第二，这里所有的加法为半加，无进位，因而除法中所有减法亦为半减，亦无借位。此时，0-1 和 1-0 一样，结果都为 1，且不向高位借位。第三，上述除式中，我们未写出商，因为我们关心的是余数，商为多少是无关紧要的。

上面产生的码字是问题 C ～E 供选择答案中的⑥。不难看出用正确方式产生的码字再被生成多项式 G（X）的系数中除得到的余数必为 0，即可以整除。若将此码字存入存储器或发往远地，若再读了或接受时码字不满足上述规律，则必然在存取或传输中出现了差错。这就是循环冗余的检错觉功能。考查 C～E 供选择答案中的其余各码字，比如说①，因为：



能整除，故它不是出错的码字。

再看 2，因为不能整除，故该码字有错。码字 3 和码字 6 有相同的信息位（前 8 位），而后 5 位冗余不同，既然 6 是正确的码字，则 3 必然是错误的码字。这样，我们已经找到了题目所要求的两个错误码字。实际上，若我们将码字 4 或 5 用 110011 去除，可以看到都能被其整除，即它们都不是出错的码字。



[答案]A：④ B：③ C：⑥ D：② E：③

试题 6 （1994 年试题 8）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

某机器中码字长 15 位（包括信息位和海明校验位），采用了可纠正一位错的海明校验。识码字位从左到右用 b1,b2…b15 编号，其海明校验方程式为

b1⊕b3⊕b5⊕b7⊕b9⊕b11⊕b13⊕b15=0

b2⊕b3⊕b6⊕b7⊕b10⊕b11⊕b14⊕b15=0
b4⊕b5⊕b6⊕b7⊕b12⊕b13⊕b14⊕b15=0
b8⊕b9⊕b10⊕b11⊕b12⊕b13⊕14⊕b15=0

若在答案中给出的码字最多只有一位错，请找出无错的识字 A ；第 2 位 b2 错的码字 B ； b4 错的码字 C ； b6 错的码字 D 和 b8 错的码字 E 。

供选择的答案：

A～E ①000100101101010 ②010100101100010 ③010100111101010
④010101101101010 ⑤001111010000111 ⑥001111010001111
⑦101111010000111 ⑧001011010000111

[解析]

从题中给出的海明校验的方程式可知：

（1） 若码字无错，则将其相应位代入 4 个校验方程式的左边，计算后应有尽有全为"0"；

（2） 若只是一位错，仔细观察校验方程式的构成可见，b1 只出现在第 1 个校验方程中，因而会使其值由"0"变为"1"，而其余 3 个方程的值仍为"0"，若将 4 个校验议程的值由下至上排列为 0001，正好是二进制值 1，同样， b2 只出现在第 2 个校验方程中：

（3） 若只有 b2 值，则 4 个校验方程的值由下至上排列后为 0010，正好是十进制值 2。我们可得到 bi(I=0,2,…,15)一位错，代入校验方程左边求值，并由下至上排列后正好是 I 的二进制值。
我们可分别计算各个选择答案的校验方程的左边的值：

b1=0,b2=0,b3=0,b4=0,b5=0,b6=0,b7=0,b8=0,b9=0,b10=0,b11=0,b12=0,b13=0,b14=0,b15=0,分别代入 4 个校验方程的左边得到：

0⊕0⊕0⊕1⊕1⊕0⊕0⊕0=0
0⊕0⊕0⊕1⊕1⊕0⊕1⊕0=1
1⊕0⊕0⊕1⊕1⊕0⊕1⊕0=0
0⊕1⊕1⊕0⊕1⊕0⊕1⊕0=0

①$(0010)_2$=2,故第 2 位 b2 错.对于其他供选择的答案可进行类似的计算得到 4 个样验方程左边的值.
②$(1100)_2$=12.第 12 位 b12 错.

③$(1000)_2$=8,第 8 位 b8 错

④$(0110)_2$=6,第 6 位 b6 错

⑤$(0000)_2$=0,无错.

⑥$(1100)_2$=12,第 12 位 b12 错.

⑦$(0001)_2$=1,第 1 位 b1 错.

⑧$(0100)_2$=4,第 4 位错 b4.

[答案]A:⑤ B:① C:⑧ D:④ E:③

试题 7 (1993 年试题 8)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应

栏内。

假设机器中存有代码 0100011.

87 1

若将该码视为海明码,其校验方程为
b1⊕b3⊕b5⊕b7=0,b2⊕b3⊕b6⊕b7=0,b4⊕b5⊕b6⊕

b7=0,经校验其出错位为第 A 位.

若将该码的第 7 位至第 4 位视为信息位,它的(7,4)循环码的生成多项式为 $g(x)=1+x+x^3$,则信息位后随冗余位构成的循环码为 B .将该码第 8 位添加偶校验后,若视为十六进制数为 C ;若视为余 3 码,对应的十进制数为 D ;若视为移码,代表的十进制数为 E .

供选择的答案

A:①2 ②4 ③5 ④6 ⑤7

B:①0100011 ②0100111 ③0100100 ④0100101 ⑤0100110

C ～E:①A3 ②B3 ③23 ④35 ⑤46
⑥53 ⑦70 ⑧73 ⑨83 ⑩C6

[解析]

若某位序号的二进制为 X3X2X1，则当 Xi=0(Xi=0 或 1,I=1,2,3)时，该位不出现在第 I 个校检方程中；而 Xi=1 时，该位出现在第 I 个校验方程。因此，若将代码的值代入 b1⊕b3⊕b5⊕b7=0、b4⊕b5⊕b6⊕b7=0 的左面，分别求出其值 x1、x2、x3，如果全为 0 则无错；如果不全为 0，则单个出错位是 X3X2X1。

对本题中具体给出的代码来说：

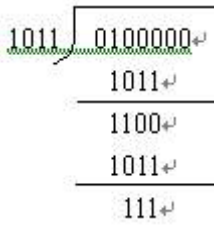
1⊕0⊕0⊕0=1=X1

1⊕0⊕1⊕0=0=X2

0⊕0⊕1⊕0=1=X3

$X3X2X1=101_{(2)}$ 即其错位为第 5 位。

若题中给出代码的第 7 至第 4 位为信息位，则 0100，其对应的信息多项式 M（X）是 X2；循环码生成多项式 g(x)=x3+x+1,其系数为 1011；冗余位的位数即生成多项式 g(x)的次数，为 3。按照循环码的法则，利用多项式系数相除可求出余式 R（x）的系数，所有减法都是"半减"，不论 1 减 0 或都 0 减 1 所得的差都是 1，无"错位"的问题，见下式：



故 R(x)的系数，即冗余位，为 111。信息位后随冗余位构成的循环码为 0100111。将题中给出代码在第 8 位添加偶校验位后成 10100011。若将它视为十六进制数则为：$10100011_{(2)}=A3_{(16)}$

若将它视为余 3 码，因 1010 对应于十进制数位 7，而 0011 对应于十进制数位 0，故其对应的十进制数为 70。若将 10100011 视为移码，它所对应的 M 进制真值为 100011，化为等值的十进制数是 35。

因此，若将机器中代码 0100011

87 1

视为海明码，其校验方程为 b1⊕b3⊕b5⊕b7=0,b2⊕b3⊕b6⊕b7=0,
b4⊕b5⊕b6⊕b7=0,经校验其出错位为第 5 位。若把该码的第 7 位至第 4 位视为信息位，它的（7，4）循环码的生成多项式为 $g(x)=1+x+x^3$，则信息位后随冗余位构成的循环码为 4。将该码第 8 位添加偶校验后，若视为十六进制为 A3；若视为余 3 码，对应的十进制数为 70；若视为移码，代表的十进制数为 35。这里指的是能纠正一位错的海明码。在无错情况下，3 个校验方程应全部满足。若从 b1 到 b7 有某位出错，必然造成校验方程不能全部满足，即校验方程左面式子的运算结果不全等于 0。

[答案]A：③ B：② C：① D：⑦ E：④

试题 8 （1992 年试题 11）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

计算机系统部线是连接器、主存、I/O 控制卡等部件的一组信息线。

例如，A、B 和 C 都是系统总线。A 是 8 位的用于工业控制领域的主流总线，已成为 IEEE961 标准： B 是与工业标准结构的 AT 总线兼容并扩展了其功能的 32 位总线； C 则是 IBM 公司推出的与 AT 总线不兼容的首先在 PS/2 机器上采用的 32 位总线。

设备接口线是 I/O 控制卡和 I/O 设备之间的一组连接线。例如，D 和 E 是两种磁盘接口的标准。E 的传输速度更高，也可支持更大的硬盘空间。

供选择的答案

A～E①ISA ②ST506 ③PS232 ④STD ⑤SCSI ⑥MCA ⑦MULTIBUS ⑧EISA

[解析]

计算机系统总线是一组用来连接计算机中央处理器（CPU）、主存和 I/O 控制卡等部件的信息线，是各部件间传送信息的公共通信，I/O 控制卡通过设备接口线 I/O 设备连接。

支持工业控制领域应用的总线有 STD、STE 和 G-64 等，通常只支持 8 或 16 位的微处理器，但 I/O 扩展能力强、组合灵活、价格低廉，它们属于低端总线。STD 总线的数据总线是 8 位的，是目前最广泛应用于工业控制领域的一种主流总线，也已成为电气电子工程师协会 IEE961 标准；STE 总线可以看成是欧洲版的 STD 总线，已成为 IEEE1000 标准；G-64 总线是另一种欧洲用户使用较广泛的适合于工业控制应用的低端总线。

另一类系统总线向提高系统的处理能力方向发展，如支持 32 位微处理器等，EISA、MCA 和 MULTIBUSI 等都是高端总线。扩展工业标准结构 EISA（Extended industrial Standard Architecture）总线是对原有的 AT 总线即 ISA 总线的扩展，但又仍保持与 ISA 兼容，它的插槽分为两段，较浅的部分是原 ISA 总线的信号线，而较深的部分是 EISA 的信号线，后一部分有一个突出的卡口。这样一来，原有的 ISA 卡可以插入较浅的部分继续使用，而新的 EISA 卡上与插槽突出卡口配合有一凹口，所以可插得列更深。其性能的扩展包括提供快速、单周期的突发传送 32 位数据通道，可达 33Mbit/s 的数据传输率，允许访问多达 8 个 DMA 控制器，可独立编程，支持 6 个级别的集中总线仲裁，可以通过软件对配置构成进行管理等方面。

MCA 是微通道结构（Micro Channel Architecture）总线，在电气性能和机械性能方面都与原有的 AT 总线不兼容，不支持任何现存的 ISA 卡。MULTIBUS 只支持 8 位和 16 位微处理器，为了适应 32 位微处理器的要求，Inter 等公司后来又联合推出第二代的总线 MULTIBUSII.

设备接口线是 I/O 控制卡和 I/O 设备之间的连接线。ST（Standard Type）506 和 SCSI（Small Computer System Interface）是两种可用于连接磁盘设备的接口标准。其中，ST506 主要用于 5。25 英寸的盘上，它是 IBMPC/XT 机的标准硬盘接口。SCSI 在超级微机、工作站和小型计算机中广泛采用，其数据传输率要比 ST506 高，它可支持的最大盘空间也比 ST506 大。RS232 则是一种串行设备接口线的标准，通常用来连接终端和远程通信设备，不能用于连接磁盘设备。

[答案]A：④ B：⑧ C：⑥ D：② E：⑤

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

根据"冗余校验"的思想,码距可用来判断使校验码制冗余的程度,并估价其查错、纠错能力。"8421"码的码距为 A ，因而它 B 。若一组海明（Hamming）码有效信息位 k=4，校验位 r=3，则其码距为 C ，用它能够发现 D 位错，并可纠正 E 位错。

供选择的答案

A、C、D、E：①0 ②1 ③2 ④3 ⑤4 ⑥7

B：①能发现 1 位错 ②能纠正 1 位错 ③能发现并纠正 1 位错 ④不能查错、纠错

[解析]

码距是两个码字间不同位的个数，如 1010 和 1100 中间两位不同，两码字的码距为 2。一个码字集的码距则指该码字集中所有码字对码距的最小值。"8421"码字集中许多码字对（如 0110 和 0111）的码距都是 1，故该码的码距为 1。

"冗余校验"的基本思想是：对于任意给定的信息位，按照一定的编码规则，可求得确定的冗余校验位，一起构成该编码的合法码字。可以证明：一种"冗余校验"码合法码字集的码距离若为 2d+1，则它能够发现 2d 位错，并能纠正 d 位错。因为若合法码字集的码距为 2d+1，则任何 2d 位的错，必然只可能使一个合法码字变成一个非法码字，而不可能变为另一个合法码字，因而能被检测出来。进一步，由于合法码字集的码距为 2d+1，那么任何一个合法码字由于 d 位错而产生的非法码字子集和另一个合法字由于 d 位错而产生的非法码字通过判断它属于哪一个子信息位数 k 和校难验位数 r 间满足海明不等式 $2^r \geqslant k+r+1$

本题中给出 k=4,r=3，恰好满足该不等式的最低要求（取等号），因而用它能发现 1 位错，并可纠正 2 位错，码距为 3。

[答案]A：② B：④ C：④ D：③ E：②

试题 10 （1991 年试题 90）
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的对应栏内。

为了快速传送大量数据，微型计算机中采用存储器直接访问技术，简称 DMA。用 DMA 方式传送进，在存储器和 A 之间直接建立高速传输数据的通路，不需要 B 的干预。利用 DMA 方式传送数据时，数据的传送过程完全由称为 DMA 控制器的硬件控制。DMA

（1） 向 CPU 申请 C 传送。

（2） 在 CPU 允许 DMA 工作时，处理总线控制的转交。

（3） 在 DMA 期间管理 D ，控制数据传送。

（4） 确定数据传送的起始地址和 E ，并在传送过程中不断修正。

（5） 数据传送结束时，给出表示 DMA 操作完成的信号。
供选择的答案

A～E:①控制台 ②硬件 ③外部设备 ④数据长度 ⑤CPU ⑥存储器 ⑦DMA ⑧系统总线 ⑨数据方向 ⑩传输速率

[解析]

直接存储器访问 DMA(Direct Memory Access)是一种不需要 CPU 干预,在存储和外部设备之间直接通过系统总线高速传输数据的方法.在此情况下数据的传送过程完全由 DMA 控制器控制,其过程如下:由 DMA 期间由 DMA 控制器控制,其过程如下:由 DMA 控制器向 CPU 发出 DMA 控制器管理系统总线,控制数据传送,确定数据传达室送的起始地址和传送的数据长度 ,并在每个传送一个数据单元后,对地址和长度分别作增加或减少的相应调整;当长度值减为零后数据传送结束,由 DMA 控制器给出表示 DMA 操作完成的信号,重新由 CPU 接管对系统总线的控制.

用 DMA 方式传送时,在存储器和外部设备之间直接建立高速传输数据的通路,不需要 CPU 的干预.利用 DMA 方式传送数据,数据的传送过程完全由称为 DMA 控制器的硬件控制.DMA 控制器具有如下功能:向 CPU 申请 DMA 传送;在 CPU 允许 DMA 工作时,处理总线控制的转交;在 DMA 期间管理系统总线,控制数据传送;确定数据传送的起始地址和数据长度,并在传送过程中不断修正;数据传送结束时,给出表示 DMA 操作完成的信号.

[答案]A:③ B:⑤ C:⑦ D:⑧ E:④

试题 11 (1990 年试题 7)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号定在答卷的
对应栏内。

1. 查找时间是 A

2. 光盘可以极大地提高 B .

3. 微型计算机常配的滚筒式绘图机中 C.

4. 与激光打印机有关的概念是 D

5. 阅读条码的硬件设备是 E.

供选择的答案

A: ①使磁头移动到要找的柱面上所需的时间

②在柱面上找到要找的磁道所需的时间

③在磁道上找到要找到的扇区所需的时间

④在扇区中找到要找的数据所需的时间

B:①可移动性 ②传送速率 ③奇偶校验能力 ④存储容量

C:①只能配一支绘图笔. ②绘图笔沿两条坐标轴运动.
③绘图笔沿一条坐标轴运动;图纸沿另一条坐标轴运动
④图纸沿两条坐标轴运动.

D:①光纤、聚焦、折射 ②曝光、显影、定影
③光笔、点阵、扫描 ④光栅、映像、合成

E：①读卡机 ②光扫描器 ③光符阅读器 ④磁条阅读器

[解析]

1. 查找时间（Seek Time）就是磁头沿切线方向移动到要找的磁道或柱面所需的时间，在磁道上找到查找目的扇区所需的时间是等待时间（Latency Time）.

2. 光盘可以极大地提高存储容量。但并不一定能极大地提高可移动性、传送速率及奇偶校验能力。

3. 滚筒式绘图机的绘图笔沿着滚筒的轴向作横向运动；图纸随着滚筒滚动，相对于绘图笔作纵向运动,两个方向运动的合成使绘图笔在图纸上画出的需的各种图形。这种绘图机通常也配有多支绘图笔。

4. 激光打印机的工作原理：计算机输出的信息控制激光图像发生器，形成激光图像照射到感光鼓鼓面上。鼓面上未被光照部分保留原有充电电荷，而被光照射的部分原充电电荷消失，这样使鼓面曝光，形成静电潜像，已有潜像的鼓面转入显像器时，由于显像器内有色粉，能靠静电吸引力吸附在已曝光的鼓面上，将静电潜像显影为色粉图像。然后，再将色粉图像转印到记录纸面上，通过加热使色粉熔化定影在纸面上，形成印刷拷贝。因此，供 D 选择的答案②中的曝光、显影和定影都是和激光打印机有关的概念。

5. 光扫描器是阅读条形码原硬件设备,读卡要是用来穿孔卡上的信息读入计算机.光符阅读器是读入字符的一种计算机输入设备.

[答案]A: ① B:④ C:③ D:② E:②

试题 12 (1990 年试题 8)

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.用二进制加法器对二-十进制编码的十进制求和,当和的 4 位二-十

进制编码(相当于一位十进制数)小于等于 1001(相当十进制数 9)且向高位无进位进, A ;当和小于等于 1001 且向高位有进位时, B ;当和大于 1001 时, C .

按照国家《信息交换用汉字编码字符集--基本集》（即 GB2312）规定，一个汉字由 D 个字节组成。为了达到中西兼容的目的，区分汉字与 ASCII 码，汉字编码的最高位为 E 。

供选择答案

A、B、C：①不需修正 ②必须进行减 6 修正
③必进行加 6 修正 ④修正方法不确定

D、E：①0 ②1 ③2 ④2.5 ⑤3 ⑥4

[解析]

用二进制加法器对于二-十进制编码的十进制求和，当和小于等于 1001（十进制数的 9）且向高位无进位时，不需修正，例如：

```
    0011
+)  0011
------
    0110
```
对应于十进制运算
```
    3
+)  3
------
    6
```

当和不于、等于 1001 且向高位有进位时，则必须进行修正即再加上 6，我们称为"加 6"修正，才会得到正确的结果。例如：

```
    1000
+)  1000
------
   10000
+)   0110
------
   10110
```
对应于十进制运算
```
    8
+)  8
------
   16
```

当和大于 1001 时，也要进行加 6 修正。例如：

```
    0110
+)  0110
------
    1100
+)  0110
------
   10010
```
对应于十进制运算
```
    6
+)  6
------
   12
```

根据国家标准 GB2312 规定，在《信息交换用汉字编码字符集--基本集》中，一个信息交换用汉字由 2 个字节组成，为了达到中西文兼容的目的，区分汉字与 ASCII 码，规定汉字编码的最高位为 1。

[答案]A：① B：③ C：③ D：③ E：②

3.2 存储器系统

3．2．1 主要知识点

掌握各类存储器的功能、特性和使用，重点把握虚拟存储器、Cache（高速缓冲存储器）和多级存储器的有关内容。

3．2．1．1RAID 存储器

廉价磁盘冗余阵列 RAID，用多个较小的磁盘驱动器替换单一的大容量磁盘驱动器，同时合理地在多个磁盘上分布存放数据以支持同时从多个磁盘进行读写，从而改善了系统的 I/O 性能。RAID 机制中共分 6 个级别，工业界公认的标准分别为 RAID0～RAID5，其共同特征是：①RAID 由若干个物理驱动器组成，但对操作系统而言仍是一个逻辑驱动器；②数据分布在阵列中多个驱动器上；③冗余的磁盘容量用以保存奇偶信息，以便在磁盘失效时进行恢复（只有 RAID0 不支持该特征）。

RAID 应用的主要技术有：①分块技术。对主机请求读定的数据进行分块，使之分布于多台磁盘上。②交叉技术。对存放在多台磁盘上数据的读写，采取交叉方式进行。③重聚技术。对多台磁盘上下班存储空间进行重新编址，使数据按重新编址后的空间进行存放。

RAID0 级（无冗余和无校验的数据分块）：具有最高的磁盘空间利用率，易管理，但系统的故障率高，属于非冗余系统。

RADI1 级（磁盘镜像阵列）：由磁盘对组成，每一个工作盘都有其对应的镜像盘，上面保存着与工作盘完全相同的数据拷贝，具有最高的安全性，但磁盘空间利用率只有 50%。

RAID2 级（采用纠错海明码的磁盘阵列）：采用了海明码纠错技术，用户需增加校验盘来提供单纠错和双验错功能。对数据的访问涉及到阵列中的每一个盘。大量数据传输出时 I/O 性能较高，但不利于小批量数据传输，实际应用中很少使用。

RAID3 和 RAID4 级（采用奇偶校验码的磁盘阵列）：把奇偶校验码存入在一个别独立的校验盘上。如果一个盘失效，其上的数据可以通过对其他盘上的数据进行异或运算得到。读者数据很快，但因为写入数据时要计算校验收位，速度较慢。

RAID5（无独立校验盘时的奇偶校验磁盘阵列）：与 RAID4 类似，但没有独立的校验盘，校验信息分布在组内所有盘上，对于大、小批量数据读写性能都很好。

RAID4 和 RAID5 使用了独立存取技术，阵列中每一个磁盘都相互独立地操作，所以 I/O 请求可以并行处理。

3．2．1．2Cache 存储器

通常在CPU和主存储器之间设置小容量的高速存储器Cache.Cache容量小但速度愉，主存储器速度较低但容量大。，通过优化调度算法，系统的性能会大大改善，其存储系统容量与主存相当，而访问速度近似 Cache。在计算机的存储系统体系中，Cache 是访问速度最快的层次。

使用 Cache 改善系统改性能的依据是程序的局部性原理，即程序的地址访问流有很强的时序相关性，未来的访问模式与最近已发生的访问模式相似。依据局部性原理，把主存储器中访问概率高的内容存放在 Cache 中，当 CPU 需要读取数据时就首先在 Cache 中查找是否有所需内容。如果有则直接从 Cache 中读取；若没有再从主存中读取该数据，然后同时送往 CPU 和 Cache。

Cache 存储器的组织结构与主存储器不一样，它以行作为基本单元/每一行又分为标志和数据域两部分，数据域中存放着若干项数据，而标志项是是这一块数据的地址标识。

当 CPU 发出对存储器的读命令后，其访问地址先送给 Cache 控制器，Cache 检查其地址标识符目录以确定是否匹配项。若发现匹配项（命中），则根据其访问地址确定是对该行数据埠中的第几项进行读取，然后该项即进入 Cache 的数据寄存器。如果没有命中，则到主存器读取数据。

当 CPU 发出访存请求后，存储器地址先被送到 Cache 控制器以确定所需数据是否已在 Cache 中，若命中则直接对 Cache 进行访问，这个过程称为 Cache 的地址映射（Mapping）。常见映射方法有直接映射、相联映射和组成映射。

当 Cache 存储器产生了一次访问合职之后，相应的数据应同时读入 CPU 和 Cache。但是当 Cache。但是当 Cache 已存满数据后，新数据和须淘汰 Cache 的地址映射（Mapping）。常见的映射方法有直接映射、相当映射和组相联映射。

当 Cache 存储器产生了一次访问未命中之后，相应的数据应同时读入 CPU 和 Cache。但是当 Cache 已存满数据后，新数据必须淘汰 Cache 中的某些旧数据。最常用的淘汰算法有随机淘汰法、先进先出法（FIFO）和近期最少使用淘汰法（LRU）。

3．2．1．3 虚拟存储器

计算机里的程序和数据通常都存放在外存储器（辅助存储器）上，直到 CPU 需要的才调入到主存储器中。虚拟存储系统的作用是给程序员一个更大的"虚拟"的存储空间，其容量可远远超过主存储器的容量，而与辅助存储器容量相当。

在使用虚拟存储器体系的系统中，由程序（CPU）使用的访存地址称为虚拟地直，程序（CPU）直接访问的存储空间称为虚拟地址空间，而主存储器的地址则称为物理地。通常虚拟地直空间远远大小主存储器容量。

3．2．2 试题解析

在硬件基础知识部分，存储器系统始终是考查的重点。从历年试题统计（见表3-3）来看，考查的内容主要集中在Cache、虚拟存储器和RAID磁盘阵列。

表3-3 历年高级程序员级存储器系统试题统计

试题	考查知识点
1992年试题10	Cache存储器与虚拟存储器
1993年试题10	辅助存储器，涉及软盘、光盘和磁带机
1994年试题7	RAID磁盘阵列
1996年试题8	Cache存储器
1997年试题9	SCSI接口，RAID磁盘阵列
1998年试题8	Cache存储器
1999年试题9	内存种类
1999年试题10	虚拟存储系统
2000年试题10	存储器有关的计算

试题1 （2000年试题10）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

假设某计算机具有1MB的内存（目前使用计算机往往具有64MB以上的内存），并按字节编址，为了能存取该内存各地址的内容，其地址寄存器至少需要二进制A 位。为了使4字节组成的字能从存储器中一次读出，要求存放在存储器中的字边界对齐，一个字节的地址码应 B 。若存储器周期为200ns，且每个周期可访问4个字节，则该存储器带宽为 C bit/s。假如程序员可用的存储空间为4MB，则程序员所、用的地址为 D ，而真正访问内存的地址称为 E 。

供选择答案

A：①10 ②16 ③20 ④32

B：①最低两位为00 ②最低两位为10 ③最高两位为00 ④最高两位为10

C：①20M ②40M ③80M ④160M

D：①有效地址 ②程序地址 ③逻辑地址 ④物理地址

E：①指令地址 ②物理地址 ③内存地址 ④数据地址

[解析]

这是一道关于内存的计算题。

1M=2^{20}，故 1MB 内存按字节编址（即寻找空间为 1M），地址寄存器至少需要 20 位。

如果采用字节编址，4 字节一次读出，即字长为 32 位，每个字有 4 个单独编址的存储字节，字地址是该字高位字节的地址，总是等于 4 的倍数，正好用地址码的最低两位（为 0）来区分同一字的 4 个字节。

若存储周期为 200ns，每个周期可访问 4 个字节，其带宽为：

（$1/200\times10^{-9}$）×4=20×10^{6}（字节/秒）=160M（位/秒）

因为可用的 4M 内存空间超大型出了实际的物理内存 1M，称为逻辑地址，实际访问内存的地址为物理地址。

[答案]A：③ B：① C：④ D：③ E：②

试题 2 （1999 年试题 9）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

用作 存储器的芯片有不同的类型。

可随机读写，且只要不断电则基本存储信息就可一直保存的，称为 A 。

可随机读写,但即便在不断电的情况下其存储的信息也要定时刷新才不致丢失失的，称为 B 。

所存信息由生产厂家用来掩膜技术写好后就无法再改变的称为 C 。

通过紫外线照射后可擦除所有信息，然后重新写入新的信息并要可多次进行的，称为 D 。

通过电信号可在数秒钟内快速删除全部信息，但不能进行字节级别删除操作的。称为 E 。

供选择答案

A、B：①RAM ②VRAM ③DRAM ④SRAM

C、D：①EPROM ②PROM ③ROM ④CDROM

E： ①E2PROM ②Flash Memory ③EPROM ④Virtul Memory

[解析]

本题考查计算机存储器知识，比较容易。存储器是用来存放程序如数据的，因此对它的要求是既能写入又能读入，这样存储器称为随机存储器（RAM）。目前计算机中广泛采用的是金属氧化物半导体（MOS）存储器，随机存储器在停电的情况下会丢失所存储的信息。

随机存储器分为静态与动态两种。静态存储器（SRAM）的内容，在不停电的情况下通史长时间保留不变；动态存储器（DRAM）的内容光焕发，即使不停电的情况下，隔一定时间以后也会自动消失，因此在消失前要根据原来内容重新写一遍，这就是再生和刷新，一般以 2ms 为周期进行刷新。

另外,根据是否可以写入内容,又可以分为随机存储器(RAM)和只读存储器(ROM)。通常工厂提供的只读存储器（掩膜 ROM）在出厂时已经写好内容，还有一种只读存储器，在出厂时芯片内没有内容。允许用户使用特殊的仪器写入内容，然后安装到计算机上，但在计算机运行过程中却不允许写入内容，这叫做可编程的只读存储器（PROM）。

在可编程只读存储器中，有些是在写入内容后就无法改变其内容的，而有些却是可以通过特殊的仪器将原先的内容擦掉再重新写入的，这种 PROM 叫做可擦除只读存储器，（EPROM），在 EPROM 中，有一种可以通过电来擦除内容的 EPROM，称为电擦除可编程只读存储器（EEPROM 或者 E2PROM）。

至于 Flash Memory,通常称为闪存，通过电信号可以在数秒钟内快速删除全部信息，但不能进行字节级别删除操作。

[答案]A：④ B：③ C：③ D：① E：②

试题 3 （1999 试题 10）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

虚拟存储器的作用是允许 A 。它通常使用 B 作为它一个主要组成部分。对它的调度方法与 C 基本相似，即把要经常访问的数据驻留在高速存储器中，因为使用虚拟存储器，指令招待时 D 。在虚拟存储系统中常使用相联存储器进行管理，它是 E 寻址的。

供选择的答案

A：①直接使用外存替内存

②添加此地址字长允许的更多内存容量

③程序直接访问比内存更大的地址空间

④提高内存的访问速度

B：①CDROM ②硬盘 ③软盘 ④寄存器

C：①Cache ②DMA ③I/O ④中断

D：①所需的数据一定能在内存中找到 ②必须先进行"虚"、"实"地址交换
③必须事先使用覆盖技术 ④必须将常用子程序先调入内存

E：①按地址 ②按内存 ③寄存器 ④计算

[解析]

本题主要考查虚拟存储系统。在同一个作业内部，作业进程能够得到的存储空间会小于某些作业的地址空间，为使这样的作业也能够在系统中运行，计算机系统使用了虚拟存储技术。所谓虚拟存储技术，即在内存在中保留作业/进程的一部分程序或数据，在外存中放置整个地直空间的副本。作业运行过程中可以随机访问内存中的数据或程序，但需要的程序或数据不在内存时，就将内存中部分内容根据情况写回外存，然后从外存调入所需程序或数据，实现作业内部的局部对换技术，从而允许作业的地址宽间大于实际分配的存储区域。

虚拟存储技术可以是基于手段式的，可以基于页式的，也可以是基于段页式的。影响虚存性能的关键是命中率，即一次操作中其对象在内存的概率。虚拟存储之所以在行，关键在于程序具有局部性。程序局部性有两个方面的含义：（1）时间局部性，如果一条指令被执行，则在不久以后可能再次被执行；（2）空间局部性，一段时间里程序所使用的操作地址相对集中在较小的范围内。

虚拟存储技术常用的页面调度技术是请求式页面调度,即除了在页表中存放逻辑页呈与物理页号的对应关系等数据外，还要标训该页是否在内存。当一条指令的操作对象所在页不在内存时，发出缺页中断，转入操作系统处理。这时系统按照某种淘汰算法挑选某一物理页，根据情况决定是否将其写回外存，然后从外存在中调入相应页面覆盖之，修改页表，缺页中断逻辑地址，而内存中存储的数据和程序是以物理地址来表示的，这样，在执行一条指令时，必须先进行从逻辑地址的交换，即地址映射。

Cache 即高速缓冲存储器。内部 Cache 对程序操作是透明的，而且不需要使用外部总线。通常,Cache 中存放一部分内存的内容，当执行读操作时，首先查找地址是否存在于 Cache 中，如果存在，就立即从 Cache 读取，否则从内存中读取，同时将内存中该部分内容写入 Cache，以便今后可以直接读取 Cache。这与虚拟存储技术的页面调度方式相似。
[答案]A：③ B：② C：① D：② E：③

试题 4 （1998 年试题 8）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

设有 3 个指令系统相同的处理机 X、Y 和 Z，它们都有 4kB 的高速缓冲存储器（Cache）T 和 32MB 的内容，但是其存取周期都不一样，见表 3-4（Tic 和 Tim 分别表示 I 处理机 Cache 存取周期和主存存取周期）。

表 3-4 处理机存取周期

处理机 / 存取周期	X	Y	Z
Tic	40ns	100ns	120ns
Tim	1μs	0.9μs	0.8μs

若某段程序，所需指令或数据在 Cache 中取到的概率为 P=0.5，则处理机 X 的存储器平均存取周期为 A us。假定指令招待时间与存储器的平均存取周期成正比，则此时 3 个处理机执行该段程序由快到慢的顺序为 B 。

若 P=0.65 时，则顺序为 C。

若 P=0.8 时，则顺序为 D。

若 P=0.85 时，则顺序为 E。

供选择的答案

A：①0.2 ②0.48 ③0.52 ④0.6

B～E:①X、Y、Z ②X、Z、Y ③Y、X、Z ④Y、Z、X
⑤Z、X、Y ⑥Z、Y、X

[解析]

在高速缓冲存储器（Cache）与主存构成的存储体系中，存储器的平均存取周期为 T=Tic*P+(1-P)*Tim，其中 Tic 为 Cache 的存取周期，Tim 为主存存取周期，P 为 Cache 的命中率，即指令或数据在 Cache 中取到的概率。具体计算见表 3-5。

表 3-5 1998 年试题 8 单位：μs

P / 处理机	X	Y	Z	快慢次序
0.5	0.52	0.50	0.46	Z、Y、X
0.65	0.376	0.38	0.358	Z、X、Y
0.8	0.232	0.26	0.256	X、Z、Y
0.85	0.184	0.22	0.222	X、Y、Z

[答案]A：③ B：⑥ C：⑤ D：② E：①

试题 5 （1997 年试题 9）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

SCSI 是一种通用的系统级标准输入/输出接口,其中 A 标准的数据宽度为 16 位,数据传送率达 20MB/s。

大容量的辅助存储器常采用 RAID 磁盘阵列。RAID 的工业标准共有 6 级。其中 B 是镜像磁盘阵列，具有最高的安全性； C 是无独立校验盘的奇偶校验码磁盘阵列； D 是采用纠错海明码的磁盘阵列；E 则是既无冗余也无校验的磁盘阵列，它采用了数据分块技术,具有最高的 I/O 性能和磁盘空间利用率,比较容易管理,但没有容错能力。

供选择的答案

A：①SCSI-I ②SCSI-II ③FAST SCSI-II ④FAST/WIDE SCSII-II

B～E：①RAID0 ②RAID1 ③RAID2 ④RAID3 ⑤RAID4 ⑥RAID5

[解析]

SCSI（Small Computer System Interface，小型计算机系统接口）标准的性能见表 3-6。

表 3-6 SCSI 接口性能参数

名称	数据宽度	数据传输率
SCSI-I	8 位	5Mbit/s
SCSI-II	16 位	10Mbit/s
FAST SCSI-II	16 位	20Mbit/s
FAST/WIDE SCSI-II	32 位	40Mbit/s

因此数据宽度为 16 位，传输率为 20MB/s 的输入/输出接口为 FASTSCSI-II。

RAID（Redundent Array of Inexpensive Disks）磁盘阵列的工业标准由 RAID0～RAID5,共 6 级，其特点如下：

RAID0 无冗余和无效验的数据分块磁盘阵列;

RAID1 镜像磁盘阵列，具有最高安全性;

RAID2 采用纠错海明码的磁盘阵列;

RAID3 和 RAID4 采用奇偶检验码的磁盘阵列（有独立校验盘）;

RAID5 无独立校验盘的奇偶检验码磁盘阵列。

[答案]A：③ B：② C：⑥ D：③ E：①

试题 6 （1996 年试题 9）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

在多级存储系统中，Cache 处在 CPU 和主存之间，解决 A 问题。若 Cache 和主存的存取时间分别 T1 和 T2，Cache 的命中率为 H，则计算机实际存取时间为 B 。当 CPU 向存储器执行读操作时，首先访问 Cache，如命中，则从 Cache 中取出指令或数据，否则从主存中取出，送 C ；当 CPU 向存储器执行写操作时，为了使 Cache 内容和主存的内容保持一致，若采用 D 法，同时写入 Cache 和主存。由于 Cache 和主存。由于 Cache 容量比主存容量小，当 Cache 已写满，但要把主存信息写入 Cache 和主存。由于 Cache 容量比主存容量小，当 Cache 已满，但要主存信息写入 Cache 时，就要淘汰 Cache 中已有信息。为了提高 Cache 的命中率，常采用一种 E 替换算法。

供选择的答案

A：①主存容量扩充 ②主存和 CPU 速度匹配 ③多个请求源访问主存在 ④BIOS 存放

B:①HT1+T2 ②(1-H)T1+HT2 ③T2-HT1 ④HT1+(1-H)T2

C:①Cache ②CPU ③Cache 和 CPU ④Cache 或 CPU

D:①写回 ②写通 ③映照 ④特征

E;①LRU ②FIFO ③FILO ④RANDOM

[解析]

计算机存储体系一般分成 3 个层次:Cache、为第 1 层次，存取速度最快，但存储容量最小，价格也最高；辅助存储器容量最大，速度最低，价格也最便宜，处于第 3 层次；主存储器居中，无论从存取速度、容量或价格等方面考虑均处于两者之间。

由于集中电路的工艺的发展，CPU 的速度越来越快，而相对来说，存储器速度跟随不上 CPU 的发展，因此阻碍了计算机整体性能的提高，为了解决主存和 CPU 速度匹配问题，在 CPU 与主存储器之间设置了 Cache。Cache 的速度是主存的数倍，基本上与 CPU 匹配。因此本题的答案 A 应选择②。

CPU 取指令或取数据时，首先检查要访问单元的内容是否已在 Cache 中，若是，则从 Cache 中读取；若不是，则到主存中读取，并将取出的内容同时送往 CPU 与 Cache 和从主存取读出操作后，能保证 Cache 和主存内容的一致性。Cache 的命中率为 H，即从 Cache 从主存读取内容的概率分别为 H 和 1-H，由此计算出平均存取时间为 HT1+（1-H）T2。所以 B 选④,C 选③.

Cache 中的内容只是主存相应部分的一个副本，当 CPU 该部分的某个单元执行写操作时，为了使 Cache 内容与主存的内容保持一致，通常采用两种办法，即"写回"和"写通"。如果在执行写操作时，数据同时写入 Cache 和主存称为写通（Write-through），该方法操作简单，可随时保持 Cache 和主存数据的一致性，但可能增加多次不必要的写入。如要向 Cache 某一单元写入多次，也要向主存相应单元写入多次，另一办法称之为"写回"（Write-back），执行写操作时一般只写入 Cache，只有当该 Cache 单元被替换(或多道程序转换)时,才将修改过的内容写回主存,这种方式写入极快.因此本题答案 D 应选择②"写通"

在供选择的答案 E 中 LRU(Least Recently Used)为"近期最少使用"算法，根据页面调入的先后来淘汰页面；FIFO（First In First Out） 为"先进先出"算法，根据页面调入的先后淘汰页面，但页面调入的先后并不能反映页面的使用情况；FILO（First In Last Out）是"先进后出"算法；RANDOM 为"随机"算法。以上都是 Cache（或虚拟存储器）的替换算法。常用的是 LRU 和 FIFO 两种算法。尤其是 LRU，通常可取得比 FIFO 更高的命中率，所以 E 选①。

[答案]A：② B：④ C：③ D：② E：①

试题 7 （1994 年试题 7）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

RAID 是一种经济的磁盘冗余阵列,它采用 A 和 B 以提高数据传输率.RAID 与主机连接较普遍使用工业标准接口为 C .

假脱机（Spooling）打印与脱机打印机有相似之处，但实际上其输出结果首先送往 D 保存，然后再在适当的时候将其调出打印出来。整个过程是由 E 控制的。

供选择的答案

A、 B：①智能控制器 ②磁盘镜像 ③磁盘双工技术 ④多磁盘驱动器

C：①RS-232 ②FDDI ③SCSI ④ST506

D、E：①主存储器 ②设备控制器 ③操作系统 ④外部存储器
⑤打印机缓冲存储器 ⑥编译程序

[解析]

RAID（Redundant Arrays of Inexpensive Disks）是一种经济的磁盘冗余阵列，采用智能控制器和多磁盘控制器以提高数据传输率。RAID 与主要连接较普遍使用的工业标准接口为 SCSI。目前，RAID 的产品多是由许多"独立"的磁盘构成冗余阵列，以提高其性能和可靠性。为了提高数据传输率，RAID 采用多磁盘驱动

器和智能控制器，使得阵列中的各个磁盘可独立并行工作，减少了数据存取等待时间。

假脱机（Spooling）打印与脱机打印有相似之处，实际上其输出结果首先送往外部存储器保存，然后再在适当的时候将其调出打印。整个过程是由操作系统控制的。

RS-232 是用来连接终端或调制解调器的串行接口；FDDI（光纤分布式数据接口）是一种光纤环形网络的名称；ST506 则是个人计算机上用来连接温彻斯特硬盘的较低速的接口，不能用来连接 RAID。

[答案]A：① B：④ C：③ D：④ E：③

试题 8 （1993 年试题 10）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

软盘使用前先要进行格式化（FORMAT），是为了确定 A 。光盘存储器利用光束在记录表面读写信息，使用的是 B 。光盘当前受到用户重视是因其有下述突出优点：C 。一台高性能 9 磁道、半英寸（1 英寸=25.4mm）的磁带机，记录密度是 256B/mm,带速 4m/s ,启停时间 10ms，带长 900m，以 EBCDIC 码按块记录文件，每个记录长为 128B，块间间隔 10mm，块化系数为 16，那么整盘带的记录容量最接近 D 字节（取整数），从磁带上读出全部记录，需要 E ms 时间。

供选择答案

A：①磁道数 ②道密度 ③编码格式 ④记录格式

B：①红外光 ②紫外光 ③激光 ④可见光

C：①容量大、可靠性高、便于携带 ②容量大、可靠性高、高速
③容量大、便携带、可反复读写 ④可靠性高、高速、可反复读写

D：①160M ②400M ③100M ④60M

E：①125010 ②125000 ③215000 ④225010

[解析]

一个计算机系统上使用一张软盘，必须首先将其格式化成该系统所能识别的记录格式。为了确定软盘的记录格式，软盘使用前先要进行格式化。一张软盘片上有多少磁道，磁道密度有多少，有统一的国际标准，因而一张空白的软盘可以用在任何计算机系统上。但是软盘。不同容量的软盘，其记录格式也不一定相同。同一计算机系统中，也支持不同容量的软盘。不同容量的软盘，其记录格式也不同，如 720KB 的软盘与 1.44MB 的软盘，其磁道数就不同。

光盘存储器是租用激光束照射到记录表面，利用热作用和介质表面熔化以改变其光学反射特性或改变介质表面局部磁场方向来记录信息，而后再利用激光束照射到记录介质上，根据反向光的强弱来读出信息的。光盘因其容量大（一般容量为650MB）、可靠性高和能进行高速存取等优点。软盘虽然容量小（常用的3.5英寸软盘容量为1.44MB）、速度慢慢，但有可反复读写又极易携带的优势，因此使用方便。

磁带一般是按块存储数据的，每块中又有若干记录，每块中的记录数就称为块化系数。块与块间有块间间隔。

带记录容量

=带块数*块记录数据长度*记录密度

=(带长/(块记录长度+块间间隙))×块化系数×(记录字节数/记录密度)×记录密度

=(900m/(8mm+10mm))×16×(128Byte/(256Byte/mm))×256Byte/mm

$=5\times10^{4}\times16\times128Byte$

$=1024\times10^{5}Byte$

=100MB

从磁带上读出全部记录所需时间：

所需时间

=启停时间+带长/带速

=10ms+900m/4m/s

=225010ms

[答案]A：④ B：③ C：② D：③ E：④

试题9 （1992年试题10）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

Cache介于 A 之间，由 B 完成信息动态调度，目的是使用 C ；虚拟存储器是为了使用权用户运作比主存在容量大得多的程序，它要在 D 之间进行住处动态调度，这种调度是由 E 来完成的。

供选择答案

A、D：①CPU 和 I/O BUS ②地址寄存器和数据寄存器
③CPU 和主存 ④ 双机系统

C：①打印信息不丢失 ②主存和 CPU 速度匹配
③显示器分辨率提高 ④汉字功能增强

B、E：①软件 ②硬件 ③操作系统和硬件 ④固件
⑤BIOS ⑥操作系统

[解析]

在多次存储体系中，高速缓冲存储器 Cache 外在 CPU 和主存储器之间，其容量比主存小得多，但存取速度却比主存快得多。Cache 中的内容是主存中部分内容的拷贝，当 CPU 需访问这部分内容时就可以访问 Cache，而不必访问主存，此时称为命中；若需访问的内容不在 Cache 中（即不命中），则需从主存储器取出。Cache 和主存中信息的映象关系以及它们之间的住处调度是由硬件来实现的。Cache 的存在对编程者来说是透明的。采用 Cache 后，只要其容量足够大，并有恰当的调度算法，利用程序和数据的局部性可以获得相当高的命中率。这样就使得用起来等价于有一个速度与 Cache 相当，比主存更容易与 CPU 速度匹配，容量又与主存相当的存储器。

虚拟存储器允许用户用比主存容量大得多的地址空间来编程，以运行比主存实际容量大得多的程序。用户编程所用的地址称为逻辑地址（又称虚地址），而实际的主存地址则称为物理地址（又称实地址）。每次访问内存时都要进行逻辑地址到物理地址的转换。实际上，超过主存在实际容量的那些程序和数据是存放在辅助存储器中，当使用时再由辅存调入。地址变换以及主存和辅存间的信息动态调度是硬件和操作系统两者配合完成的。

[答案]A：③ B：② C：② D：⑤ E：③

3.3 安全性、可靠性和性能评价

3．3．1 主要知识点

了解计算机数据安全和保密、计算机故障诊断与容错技术、系统性能评价方面的知识，掌握数据加密的有关算法、系统可靠性指标和可靠性模型以及相关的计算方示。

3．3．1．1 数据的安全与保密

（1） 数据的安全与保密

数据加密是对明文（未经加密的数据）按照某种加密算法（数据的变换算法）进行处理，而形成难以理解的密文（经加密后的数据）。即使是密文被截获，截获方也无法或难以解码，从而阴谋诡计止泄露信息。数据加密和数据解密是一对可逆的过程。数据加密技术的关键在于密角的管理和加密/解密算法。加密和解密算法的设计通常需要满足 3 个条件：可逆性、密钥安全和数据安全。

（2） 密钥体制

按照加密密钥 K1 和解密密钥 K2 的异同，有两种密钥体制。

① 秘密密钥加密体制（K1=K2）

加密和解密采用相同的密钥，因而又称为密码体制。因为其加密速度快，通常用来加密大批量的数据。典型的方法有日本的快速数据加密标准（FEAL）、瑞士的国际数据加密算法（IDEA）和美国的数据加密标准（DES）。

② 公开密钥加密体制（K1≠K2）

又称不对称密码体制，加密和解密使用不同的密钥，其中一个密钥是公开的，另一个密钥是保密的。由于加密速度较慢，所以往往用在少量数据的通信中，典型的公开密钥加密方法有 RSA 和 ESIGN。

一般 DES 算法的密钥长度为 56 位，RSA 算法的密钥长度为 512 位。

（3） 数据完整性

数据完整性保护是在数据中加入一定的冗余信息，从而能发现对数据的修改、增加或删除。数字签名利用密码技术进行，其安全性取决于密码体制的安全程度。现在已经出现很多使用 RSA 和 ESIGN 算法实现的数字签名系统。数字签名的目的是保证在真实的发送方与真实的接收方之间传送真实的信息。

（4） 密钥管理

数据加密的安全性在很大程度上取决于密钥的安全性。密钥的管理包括密钥体制的选择、密钥的分发、现场密钥保护以及密钥的销毁。

（5） 磁介质上的数据加密

常用的方法有：硬加密的防复制技术、软加密的防解读技术和防跟踪技术。硬加密技术常用的 3 种方式是：①利用非标准格式的磁介质记录方式；②激光加密技术；③利用专用的硬件。

3．3．1．2 计算机系统可靠性

计算机系统的可靠性是指从它开始运行（t=0）到某时刻 t 这段时间内能正常运行的概率，用 R（t）表示。所谓失效率是指单位时间内失效的元件数与元件总数的比例，以表示，当为常数时，可靠性与失效率的关系为：

$R(t)=e^{-\lambda t}$

两次故障之间系统能正常工作的时间的平均值称为平均无故障时间 MTBF：

$MTBF=1/\lambda$

通常用平均修复时间（MTRF）来表示计算机的可维修性，即计算机的维修效率，平均修复时间指从故障发生到机器修复平均所需要的时间。计算机的可用性是指计算机的使用效率，它以系统在执行任务的任意时刻能正常工作的概率 A 来表示。

A=MTBF/（MTBF+MTRF）

计算机的 RAS 技术，就是指用可靠性 R、可用性 A 和可维修性 S 这 3 个指标衡量一个计算机系统。但实际应用中，引起计算机故障的原因除了元器件以外还与组装工艺、逻计算机可靠性模型有关。

常见的系统可靠性数学模型有以下 3 种：

① 串联系统。假设一个系统由 N 个子系统组成，当且仅当所有的子系统都能正常工作时，系统才能正常工作，这种系统称为串联系统。

设各子系统的可靠性为 R1、R2、…Rn，则整个串联系统的可靠性为：

R=2-（1-R1）（1-R2）…（1-Rn）

设各子系统的失效率为 ，则整个串联系统的失效率为：

$\lambda=\lambda 1+\lambda 2+\cdots+\lambda n$

② 并联系统。假如一个系统由 N 个子系统组成，只要有一个子系统正常工作，系统就能正常工作，这样的系统称为并联系统。

设各子系统的可靠性为 R1、R2、…、Rn，则整个并联系统的可靠性为：

R=R1×R2×…×Rn

设各子系统的失效率为 ，则整个并联系统的失效率为：

$$\mu = \frac{1}{\frac{1}{\lambda}\sum_{j=1}^{n}\frac{1}{j}}$$

③ N 模冗余系统。N 模冗余系统由 N 个（N=2n+）相同的子系统和一个表决器组成，表决器把 N 个子系统中占多数的相同结果的输出作为系统的输出。

设各子系统的可靠性均为 R。，则整个 N 模冗余系统的可靠性为：

$R = \sum_{i=N+1}^{N}\binom{j}{N} * R_0^i(1-R_0)^{N-i}$（其中 $\binom{j}{N}$ 表示从 N 个元素中取 i 个元素的组合数）

3．3．1．3 计算机系统的性能评价

性能评测的常用方法：①时钟频率；②指令执行速度；③等效指令速度法；④数据处理速率 PDR 法；⑤核心程序法。

基准程序法（Benchmark）是目前一致承认的测试性能的较好方法，有多种多样的基准程序，如主要测试整数性能的基准程序、测试浮点性能的基准程序等。

3．3．1．4 计算机故障诊断和容错

计算机的故障根据其表现出的特点，可以分为永久性故障、间歇性故障及瞬时性故障 3 类。故障诊断包括故障检测定位两个方面。

容错是采用冗余方法来消除故障影响。针对硬件，有时间冗余两种方法。主要容错技术有简单的双机备份和操作系统支持的双机容错。

3．3．2 试题解析

从历年安全性和可靠性方面的试题统计（见表 3-7）来看，主要考查系统可靠性，涉及计算机可靠性模型及相关的计算，有时与其他硬件知识类试题结合起来考查，此类试题看似复杂，其实只要复习一下相关内容，解答起来比较简单。

表 3-7 历年高级程序员级安全性与可靠性试题统计

试题	考查知识点
1990 年试题 9	涉及计算机可靠性模型，主要考查 N 模冗余系统
1991 年试题 10	系统可靠性计算
1992 年试题 8	系统可靠性计算
1996 年试题 9	系统可靠性计算
1996 年试题 11	数据加密与解密
2000 年试题 12	系统可靠性计算，海明码

试题 1（2000 年试题 12）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.
为提高数据传输的可靠性，可采用"冗余校验"的方法，海明码是常用方法之一。在此方法中，若要求能校检测出所有双位错觉，并能校正单位错，则合法码字集中的码距至少为 A 。若原始数据的字长为 5 位，则采用海明码对其样验位至少为 B 位。

对下面图 3-2（a）所示系统，仅当部件 1、部件 2 和部件 3 全部正常时系统才能政党工作，图中数字为各部分的可靠性，整个系统的可靠性近似为 C 。如果将部件 2 和部件 3 改成由两个器件构成，如图 3-2（b）所示，只要器件 a 和 b 中有一个正常就能使部件 2 正常工人，只要器件 c 和 d 中有一个正常就能使部件 3 正常工作。图中数字是各器件可靠性，则部件 2 的可靠性是 D ，整个系统的可靠性近似为 E 。

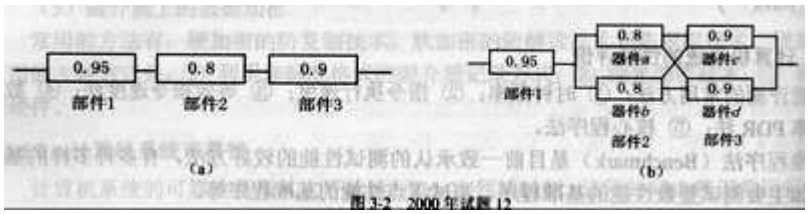


图 3-2 2000 年试题 12

供选择的答案

A：①1 ②2 ③3 ④4

B：①1 ②2 ③3 ④4

C：①0.68 ②0.72 ③0.80 ④0.92

D：①0.64 ②0.88 ③0.96 ④0.99

E：①0.82 ②0.90 ③0.94 ④0.96

[解析]

这是一道考查可靠性的综合题，除了可靠性计算之外，还涉及用于检错校验的海明码。

关于海明码问题的解答请参见 3。1。1。2 节的试题 9（1999 年试题 8）

图 3-2（ a）所示系统符合串联、系统可靠性模型，根据系统可靠性公式，求得其可靠性为：
R=R1×R2×R3=0.95×0.8×0.9≈0.68

图 3-2（b）所示系统是一个由串联和并联组合成的可靠性模型，其中部件 2 由两个并联的器件 a 和 b 构成，其可靠性为：

R2=1-（1-Ra）×（1-Rb）=1-（1-0.8）×(1-0.8)=0.96

最后与部件 1 一起计算整个系统（串联模型）的可靠性为：

R=R1×R2×R3=0.95×0.96×0.99≈0.90

[答案]A：③ B：④ C：① D：③ E：②

试题 2（1996 年试题 9）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.
设在图 3-3 和图 3-4 系统中，R1、R2、R3 为 3 个加工部件，每个加工部件的失效率均为 λ，可靠性均为 R。则图 3-3 系统的失效率为 A ，可靠性为 B 。图 3-4 中系统的失效率为 C ，可靠性为 D 。若每个加工部件的平均无故障时间为 5000 小时，则图 3-4 中系统的平均无故障时间为 E 小时。

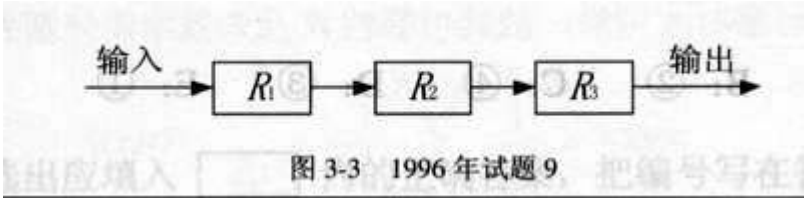


图 3-3 1996 年试题 9

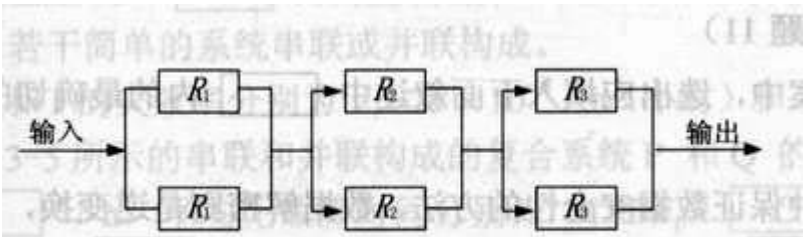


图 3-4 1996 年试题 9

供选择答案

A：①λ/3 ②λ^3 ③3λ ④1-λ^3

B：①R/3 ②R^3 ③3R ④1-R^3

C：①(3/2)λ ②(2/3)λ ③(6/11)λ④2λ

D：①（1-R^2）^{3} ②3（1-R^2） ③R^3（2-R）^{3} ④1-3（1-R^2）

E：①2500 ②5000 ③7500 ④3333

[解析]

在任意控制系统中，已知每个部件的失效率为 λ，可靠性为 Ri，对串联系统而言，系统的失效率为 λ1+λ2+λ3=3λ，问题 A 选③；系统的可靠性为 R1・R2・R3=R^3，问题 B 选②。
对并联系统（有 n 个部件，每个部件的失效率均为 λ）而言：

系统的失效率=λ/(1+1/2…+1/n)；系统的可靠性为=1-(1-R)^n。

图 3-4 所示乃串并联系统的总失效率为：

λ 系=λ 并 1+λ 并 2+λ 并 3
=λ/（1+1/2）+λ/（1+1/2）+λ/（1+1/2）=2λ，所以问题 C 选择④。

一个并联部分的可靠性=1-（1-R）^{2}=（2-R）R；

整个系统可靠性=[（2-R）R][（2-R）R]=[（2-R）R]^3=（2-R）$^3R^3$，所以问题 D 的答案为③。

由于系统的总失效率 λ 系为 2λ，且每个部件的平均无故障时间=1/λ=5000h，所以系统的平均无故障时间=1/λ 系=1/2λ=1/2・1/λ=2500h，问题 E 的正确答案为①。

[答案]A：③ B：② C：④ D：③ E：①

试题 3（1996 年试题 11）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

数据加密是一种保证数据安全性的方法，数据解密则是逆变换，即 A 。密码体制可分为 B 和 C 两大类，例如常用的 DES 属于 B ，而 RSA 则属于 C 。DES 的密钥长度为 D 位。

破密都面临多种不同的问题，其从易到难排列依次为 E 。

供选择的答案

A：①由加密密钥求出解密密钥 ②由密文求出明文
③由明文求出密文 ④由解密密钥求出加密密钥

B、C：①公开密钥 ②替代密码 ③换位密码 ④对称密钥

D：①32 ②48 ③64 ④128

E：①选择明文、已知明文、仅知密文 ②已知明文、仅知密文、选择明文
③已知明文、选择明文、仅知密文 ④仅知密文、已知密文、选择明文

[解析]

数据加密是利用加密密钥加密算法将明文（数据）转换成密文，而数据解密是利用解密密钥将密文变换成明文。所以问题 A 的答案为②。

密码体制按其对称性可分为对称密钥体制与非对称密钥体制两大类。在对称密钥体制中，加密算法之间存在一定的相依的关系，加密和解密往往使用相联系的密钥，或由加密密钥很容易推同的密钥，或由加密密钥很容易推出解密密钥；在非对称密钥体制中，有两个密钥，其中一个公开密钥，另一个为秘密密钥，因此加密密钥和解密密钥是不同的，而且很难从加密密钥导了解密密钥。DES（Data Encyrption Standard）算法采用对称密钥，DES 的密钥长度为 64 位，RSA 算法采用非对称密钥，其中包含公开密钥，所以 B 的答案为④问题 C 的答案为①，问题 D 的答案为③。

破密者，也即解密者，指的是截取到密文，而且也知道相应的明文，由此推算出用来加密的密钥或加密算法，从而解密密文。这个难度比仅知密文要小一些。

选择明文是指破密者不仅可得到密文和相应的明文，而且也可以选择被加密的明文，这比已知明文容易，因为破密者能选择特定的明文去加密，从而得到更多关于密钥的信息，继而可以更容易地推出用来加密的密钥或算法，因此问题 E 的选择答案应为①。

[答案]A：② B：④ C：① D：③ E：①

试题 4 （1992 年试题 8）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

一个复杂的系统可由若干个简单的系统串联或并联构成。

已知两个简单系统 I 和 J 的失效率分别为 λi=25×10^{-5}/h(小时)和 λj=5×10^{-4}/h（小时），则由 I 和 J 经如图 3-5 所示的串联和并联构成的复合系统 P 和 Q 的失效率分别为 λp= A /h 和 λq= B /h.平均无故障时间分别为 MTBFp= C h 和 MTBFq= D h.

系统 P 开始运行后 2 万小时内能正常运行的概率 Rp= E .



供选择的答案

A、B：①25×10^{-5} ②33×10^{-5} ③66×10^{-5} ④75×10^{-5}

C、D：①1333 ②1500 ③3000 ④4000

E：①e^{-5} ②e^{-10} ③$e10^{-5}$ ④e^{-20}

[解析]

系统的失效率 λ 指的是单位时间内的系统数与系统总数的比例。在稳定使用的阶段岢以认为 λ 是常数。系统从开始运行到某一时刻 t 这段时间内能正常运行的概率又称为系统的可靠性，是 t 的函数，记为 R（t），可以证明：$R(t)=e^{-\lambda t}$.

如果把系统故障发生的时刻看成是随机变量，则该随机变量的概率分布函数为：

F（t）=P{ε ≦t}=

1- P{ε >t}=

1- R(t)=

$1-e^{-\lambda t}$

平均无故障时间 MTBF（Mean Time Between Failures），就是从时刻 0 开始到故障发生时刻间间隔的平均数，即随机变量 ε 的平均值，可算得：

$$MTBF=\int_0^{\infty} tF'(t)dt=\int_0^{\infty}\lambda te^{-\lambda t}dt=\frac{1}{\lambda}$$

N 个可靠性分别为 Rk、失效率分别为 λk(k=1,…，n)的子系统串联构成的复合系统，只有在每个子系统都可靠时才可靠，故其可靠性 R 及失效率 λ 分别为：

R=R1R2…Rn=

$$\prod_{k=1}^{N}e^{-\lambda_k t}=e^{-\left(\sum_{k=1}^{n}\lambda_k\right)t}$$

$$\lambda=\sum_{k=1}^{n}\lambda_k$$

由此即可求复合系统 P 的失效率为：

λp=λi+λj

$=75\times10^{-5}/h$

MTBFp=1/λp≈1333h

$Rp(t)=e^{-75\times10^{-5}t}$

当 $t=2\times10^{4}$ 时有：

$Rp=e^{-15}$

对于 N 个子系统并联的情况，复合系统只有在所有子系统均失效时复合系统才失效。故有：

$$R=1-(1-R_1)(1-R_2)\mathrm{K}\,(1-R_N)$$

$$=1-\prod_{k=1}^{N}(1-R_k)$$

若 N 个子系统的失效率都是一样的，即 $\lambda k\equiv\lambda$ (k=1,…, N)则有

$R(t)=1-(1-e^{-\lambda t})N$

$F(t)=(1-e^{-\lambda t})N$

复合系统的平均无故障时间为：

$$MTBF=\int_0^{\infty}tF'(t)dt=\frac{1}{\lambda}\sum_{k=1}^{N}\frac{1}{k}$$

本题复合系统 Q 由系统 J 和两个 I 串联构在的复合系统（不妨记为 I'）再经并联构成。复合系统 I'的失效率为：

$\lambda'I=\lambda i+\lambda j=$
$5\times10^{-4}/h$
$\lambda'I=\lambda j$

N=2 时并联系平均无故障时间：

$$MTBF_q=\frac{1}{\lambda_j}\sum_{k=1}^{N}\frac{1}{k}=3000h$$

且

$\lambda q=1/MTBFq\approx33\times10^{-5}$

[答案]A：④ B：② C：① D：③ E：③

试题 5（1991 年试题 10）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

由两个相同的计算机单元组成的计算机维修双工系统，其状态转移图如图 3-6 所示。

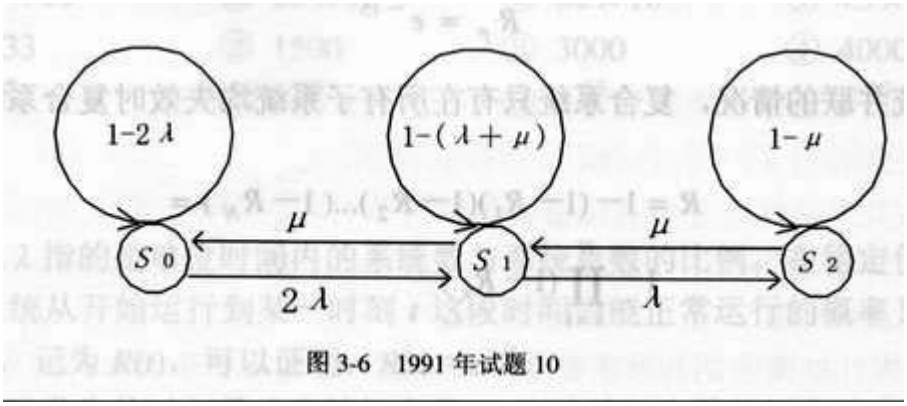


图 3-6 1991 年试题 10

其中：S0 表示两个单元均正常工作，系统有效运行；S1 表示其中一个单元正常工作，一个故障单元正在修理，系统仍有效运行；S2 表示系统失效，一个故障单元正在修理，另一个故障单元待修。λ、μ 分别表示计算机单元的故障率和修复率。该系统的状态概率转移矩阵为 A 。系统处于稳定状态时，系统无故障运行的概率为 B ；系统有效运行的概率为 C 系统失效的概率和故障的关系是 D ，和修复率关系是 E 。

供选择的答案

A：

①$\begin{bmatrix} 1-2\lambda & 1-(\lambda+\mu) & 1-\mu \\ \mu & \mu & 0 \\ 2\lambda & \lambda & 0 \end{bmatrix}$

②$\begin{bmatrix} 1-\mu & 1-2\lambda & 1-(\lambda+\mu) \\ \mu & \mu & 0 \\ 2\lambda & \lambda & 0 \end{bmatrix}$

③$\begin{bmatrix} 1-2\lambda & \mu & \mu \\ 1-(\lambda+\mu) & 2\lambda & \lambda \\ 1-\mu & \lambda & 0 \end{bmatrix}$

④$\begin{bmatrix} 1-2\lambda & 2\lambda & 0 \\ \mu & 1-(\lambda+\mu) & \lambda \\ 0 & \mu & 1-\mu \end{bmatrix}$

B, C：

①$\frac{\mu^2}{2\mu^2+2\lambda\mu+\lambda}$ ②$\frac{\mu^2}{\mu^2+2\lambda\mu+2\lambda^2}$

③$\frac{\lambda^2}{2\mu^2+2\lambda\mu+\lambda^2}$ ④$\frac{\lambda^2}{\mu^2+2\lambda\mu+2\lambda^2}$

⑤$\frac{\mu^2+\lambda\mu}{2\mu^2+2\lambda\mu+\lambda^2}$ ⑥$\frac{\mu^2+2\lambda\mu}{\mu^2+2\lambda\mu+2\lambda^2}$

⑦$\frac{\lambda^2+2\lambda\mu}{2\mu^2+2\lambda\mu+\lambda^2}$ ⑧$\frac{\lambda^2+2\lambda\mu}{\mu^2+2\lambda\mu+2\lambda^2}$

D：①随 λ 增大而减小，并生成线性关系 ②随 λ 增大而减小，但不成线性关系 ③随 λ 增大而增大，并生成线性关系 ④随 λ 增大而增大，但不成线性关系

E：①随 μ 增大而减小，并生成线性关系 ②随 μ 增大而减小，但不成线性关系 ③随 μ 增大而增大，并生成线性关系 ④随 μ 增大而增大，但不成线性关系

[解析]
设 P0、P1 和 P2 分别表示状态转移前系统处于 S0、S1 和 S2 状态的概率；P'0、P'1 和 P'2 则分别表示状态转移后系统处于 S0、S1 和 S2 状态的概率。

根据题中给出的状态转移图，写出该系统的状态转移方程如下：

P’0=（1-2λ）P0+μP1

P’1=2λP0+[1-（λ+μ）]P1+μP2

P’2=λP3（1-μ）P2

将状态转移方程写成矩阵形式为：

$$\left(P_0' P_1' P_2\right)=\left(P_0 P_1 P_2\right)\begin{bmatrix}1-2\lambda & 2\lambda & 0\\ \mu & 1-(\lambda+\mu) & \lambda\\ 0 & \lambda & 1-\mu\end{bmatrix}$$

后一个 3×3 矩阵，即为状态率转移矩阵。在系统处于稳定状态时，P’0=P0，P’1=P1，

P’2=P2，P0+P1+P2=1，可得出的系统无故障运行的概率：

$$P_1=\frac{2\lambda}{\mu}P_0$$

$$P_2=\frac{\lambda}{\mu}P_1=\frac{2\lambda^2}{\mu^2}$$

$$P_0+\frac{2\lambda}{\mu}P_0+\frac{2\lambda^2}{\mu^2}P_0=1$$

$$P_0=\frac{\mu^2}{\mu^2+2\lambda\mu+2\lambda^2}$$

系统有一个单元正常工作，另一个单元有故障正在修理，系统仍能有效动行的概率：

$$P_1=\frac{2\lambda}{\mu}P_0=\frac{2\lambda\mu}{\mu^2+2\lambda\mu+2\lambda^2}$$

总的系统可有效运行的概率为：

$$P_0+P_1=\frac{\mu+2\lambda\mu}{\mu^2+2\lambda\mu+2\lambda^2}$$

系统失效的概率为：

$$P_2 = \frac{2\lambda^2}{\mu^2 + 2\lambda\mu + 2\lambda^2}$$

由此可知，系统失效的概率和故障率的关系是随 λ 增大而增长率大，但不成线性关系；它和修复率的关系随 μ 增大而减小，但不成线性关系.

[答案]A:④ B:② C:⑥ D;④ E:②

试题 6 (1990 年试题 9)

从供选择的答案中先出填入下面关于 N 模冗余系统的叙述中的{ }内的正确答案的对应栏内.

N 模冗余系统如图 3-7 所示，由 N(N=2n+1)个相同部件的副本和一个(n+1)/N 表决器把 N 个副本中占多数的输出作为系统的输出.

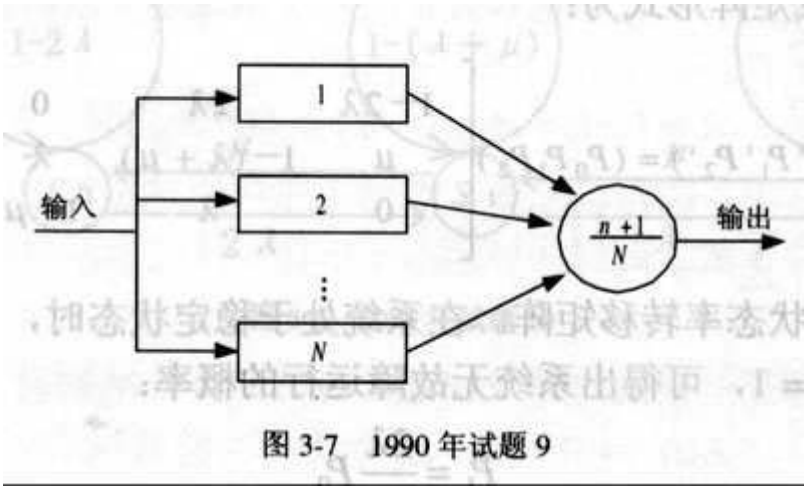


图 3-7 1990 年试题 9

设表决器完全可靠，且每个副本的可靠性为 R0，则该 N 模冗余系统的可靠性 R= A . 当 R0=$e^{-\lambda t}$，当 λ t= B ，R0= C ，R 为不依赖于 N 的恒定值 D ；当 R0 小于 C ，R 是 N 的 E .

供选择的答案

A:

①$\sum_{i=0}^{n} C_N^i R_0^i (1-R_0)^{N-i}$ ②$\sum_{i=0}^{n+1} C_N^i R_0^i (1-R_0)^{N-i}$

③$\sum_{i=n}^{N} C_N^i R_0^i (1-R_0)^{N-i}$ ④$\sum_{i=n+1}^{N} C_N^i R_0^i (1-R_0)^{N-i}$

B、C、D：①0.1 ②0.347 ③0.5 ④0.693 ⑤0.869 ⑥0.9

E:①单调递增函数 ②单调递减函数 ③对数函数 ④指数函数

[解析]

试题中图 3-7 给出的由 N（N=2n+1）个相同的副本和一个（n+1）/N 表决器组成的 N 模冗余系统中，只要有 n+1 个以上的副本正常工作，就认为系统工作正常，输出正确。因此，若假定表决器完全可靠，且每个副本的可靠性 R0，由概率论的有关理论可推出该 N 模冗余系统的可靠性为：

$$R=\sum_{i=n+1}^{n} C_N^i R_0^i (1-R_0)^{N-i}$$

若 R0=$e^{-\lambda t}$，则 R 是 λt 和 N 的函数，对于不同的 N 可画出 R 随 λt 变化的曲线簇如图 3-8 所示。

只有当 λt=Ln2=0.693 时，R0=$e^{-\lambda t}=e^{-Ln2}$=0.5

$$R=\sum_{i=n+1}^{N} C_N^i (0.5)^i (1-0.5)^{N-i}=(0.5)^N \sum_{i=n+1}^{2n+1} C_N^i=(0.5)^N \cdot \frac{1}{2}\sum_{i=0}^{N} C_N^i=(0.5)^N \cdot \frac{1}{2}(1+1)^N=0.5$$

R 为一不依赖于 N 的恒定值 0.5。

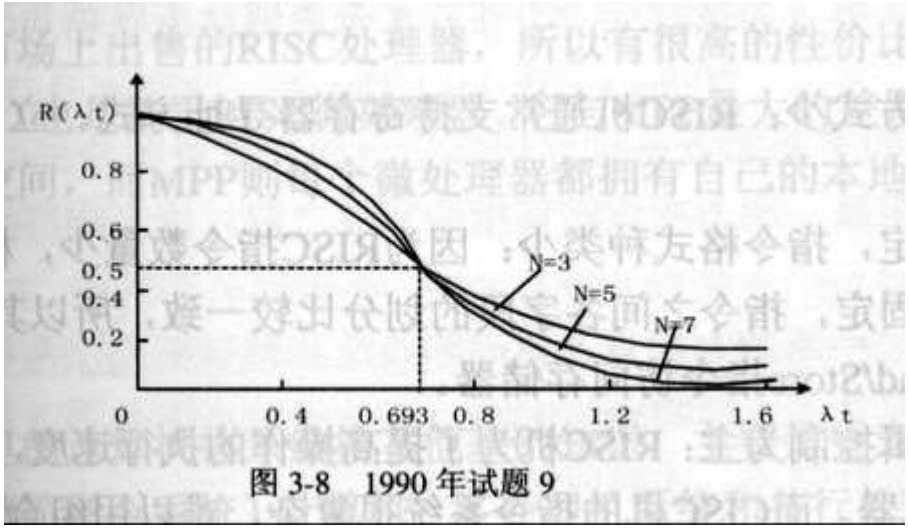


图 3-8 1990 年试题 9

当 R0 小于 0.5 时，意味着单个副本产生正确结果的可能性小于产生错误结果的可能性（I-R0），而 N 模冗余系统又是以多数副本的表决结果作为系统的输出。

显然，此时副本的个数（N）越多，就越容易引起错误，即整个系统的可靠性（R）就越小。换言之，R 是 N 单调递减函数。从上面给出的 R 依赖于 λt 变化的曲线族图中也可直观地看了，在 λt>0.693（即 R0<0.5）处，N 越大的曲线越靠下，即 R 越小。

[答案]A：④ B：④ C：③ D：③ E：②

3.4 体系结构其他基础知识

3．4．1 主要知识点

了解掌握流水线技术、RISC 计算机、并行处理技术、多处理机系统方面的基本知识。

3．4．1．1 流水线技术

流水线技术是通过并行硬件来提高系统性能的常用方法。计算机流水线技术包括指令流水线和运算操作流水线。

计算机中一条指令的执行需要若干步，通常采用流水线技术来实现指令的执行，以提高 CPU 性能。流水线设计的基本思想其实很简单，任何一个可以分解的任务都可以用流水线来做，可以设置多个处理机构，分别执行相应的子任务。为了提高流水线性能，有些处理时间长的步骤还需分解成更小的步骤，使流水线上所有步骤的处理时间相同。实际上，流水线技术对性能的提高程度取决于其执行顺序中最慢的一步。

在实际情况中，流水线各个阶段可能会相互影响，阻塞流水线，使其性能下降。阻塞主要由两种情形引起：执行转移指令和共享资源冲突。

指令流水线技术其实是把处理瓶颈从 CPU 子系统转移给了存储子系统。在存储系统中也需要使用流水线技术的 CPU 系统的处理能力。流水线计算机的存储器分成若干（4 个）独立存储体，以支持流水线方式并发访问。流水线计算机也使用了 Cache，通常分为指令 Cache 和数据 Cache，各自用于存放指令和操作数。

3．4．1．2RISC 计算机的主要特点

①指令数量少：RISC 机优先选取使用频率最高的一些简单指令以及一些常用的指令，避免使用复杂指令。

②复杂的寻址方式少：RISC 机通常支持寄存器寻址方式、立即数寻址方式以及相对寻址方式。

③指令长度固定，指令格式种类少：因为 RISC 指令数量少，格式也相对简单（与 CISC 比较）其指令长度固定，指令之间各字段的划分比较一致，所以其译码也相对容易。

④只提供了 Load/Store 指令访问存储器。

⑤以硬布线逻辑控制为主：RISC 机为了提高操作的执行速度，通常采用硬布线逻辑（组合逻辑）来构建控制器。而 CISC 机的指令系统很复杂，难以用组合逻辑电路来实现控制器，通常采用微程序控制。

⑥单周期指令执行：RISC 机因为简化了指令系统，所以很容易利用流水线技术使得大部分指令都能在一个机器周期内完成。

⑦拥有相当多的寄存器：RISC 机拥有相当数量的寄存器，可以用于存放中间数据和优化对操作数的访问，从而提高了处理器的性能。

⑧优化的编译器：RISC 的精简指令集使编译工作简单化。
采用 RISC 技术和 CPU 硬件一般具有寄存器数量多、采用流水线组织、控制器的实现采用硬布线控制逻辑电路等特点。

3．4．1．3 并行处理机

并行处理机有时也称为阵列处理机，其得名于在单一控制部件控制下的由多个处理单元构成阵列。主要用于要求进行大量高速向量或矩阵运算的应用领域。

并行处理机的并行性来源于资源重复，把大量相同的处理单元 PE 通过互联网络连接起来，在统一的控制器（CU）控制下，对各自分配来的数据并行地完成同一条指令所规定的操作。PE 是不带指令控制部件的算术逻辑运算单元。

并行处理机的特点：

① 强大的向量运算能力

② 并行方式。并行处理机的并行性来源于资源重复而不是时间重叠，利用的是并行性中的同时性而不是并发性。并行机提高运算速度的主要方法是增加处理单元的个数。

③ 适用于专门领域，如矩阵运算、向量运算等。

④ 标量运算速度对系统性能的影响也很明显。

⑤ 具有向量化功能的高级语言编译程序有助于提高并行处理机的通用性，以及减少编译时间。

3.4.1.4 多处理机

多处理机具有两个或两个以上的处理机，共享输入/输出子系统，在统一的操作系统控制下，通过共享主存或高速通信网络进行通信，协同求解一个在而复杂的问题。多处理机通过利用多台处理机进行多任务处理来提高速度，利用系统的重组能力来提高可靠性、适应性和可用性。多处理机是新一代计算机结构的基本特征，多处理机具有共享存储器和分布存储器两种不同结构。

多处理机属于 MIMD 系统，与 SIMD 的并行处理机相比，有很大的差别。其根源就在于两者的并行性等层次不同，多处理机要实现的是更高一层的作业任务间的并行。
大规模并行处理机（MPP）是由众多的微处理器（从几百到上上万）组成的大规模并行处理系统。MPP 可以采用市场上出售的 RISC 处理器，所以有很高的性价比。

3．4．2 试题分析

根据考试大纲，这里的计算机体系结构其他基础知识，主要是更为高级的硬件技术，包括流水线操作、RISC（精简指令系统）计算机、多处理机系统和并行处理。从历年的试题来看，主要考查 RISC（精简指令系统）计算机的相关知识和流水线技术。

试题 1 （2000 年试题 11）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

现采用 4 级流水线结构分别完成一条指令的取指、指令译码和取数、运算以及送回运算结果 4 个基本操作，每步操作时间依次为 60ns、100ns、50ns 和 70ns。该流水线的操作周期应为 A ns。若有一小段程序需要用 20 条基本指令完成（这些指令完全适合于流水线上执行）则得到第一条指令结果需 B ns ,完成该段程序需 C ns。

在流水线结构的计算机中，频繁执行 D 指令时会严重影响机器的效率。当有中断请求发生时，若采用不精确断点法，则将 E 。

供选择的答案

A：①50 ②70 ③100 ④280

B：①100 ②200 ③280 ④400

C：①1400 ②2000 ③2300 ④2600

D：①条件转移 ②无条件转移 ③算术运算 ④访问存储器

E：①公影响中断响应时间，不影响程序的正确执行。
④ 不仅影响中断响应时间，还影响程序的正确执行。

⑤ 不影响中断响应时间，但影响程序的正确执行。
⑥ 不影响中断响应时间，也不影响程序的正确执行

[解析]

本题主要考查流水线技术。

由流水线技术的基本特征可知，其平均时间取决于流水线最慢慢的操作，所以该流水线的操作周期为 100ns。

由题中条件可知，完成 1 条指令需要 4 个基本操作，每个操作需要 1 个周期，执行第 1 条指令时，还不能充分发挥流水线的技术优势，需要执行 4 个周期，才能得到第 1 条指令的运行结果，共需要 400ns。由于采用流水线技术，从第 1 条拽令的第 2 步基本操作开始，后续指令开始并行执行，可将 20 条指令的执行过程看作 20 段流水线，由于基本操作重叠执行，除第 1 条指令外，每条指令的执行可视为需要 1 个周期，所以总共需时间为：

400ns+(20-1)×100ns=2300ns

影响流水线性能的主要因素是执行转移指令和共享资源冲突。当流水线性执行转移指令时，会引起流水线的阻塞，因为在该转移指令完成之前，流水线都不能确定下一条提令的地址。

流水线计算机处理中断的方法有不精确断点法和精确断点法两种，采用不精确断点法，当发生中断后，计算机并不立即响应中断，而是先禁止指令再时入流水线，然后等待已在流水线中的所有指令执行完毕，才响应该中断。

[答案]A：③ B：④ C：① D：②

试题 2 （1999 年试题 12）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

计算机执行程序所需时间 P，可用 P=I • CPI • T 来估计，其中 I 是程序经编译后的机器指令数，CPI 是执行每条指令所需的平均机器周期数，T 为每个机器周期的时间。RISC 计算机是采用 A 来提高机器的速度。它的指令系统具有 B 的特点。指令控制部件的构建，C 。RISC 机器又通过采用 D 来加快处理器的数据处理速度。RISC 的指令集使编译优化工作 E 。

供选择答案

A：①虽增加 CPI，但更减少 I。 ②虽增加 CPI，但更减少 T。
③虽增加 T，但更减少 CPI。 ④虽增加 I，但更减少 CPI。

B：①指令种类少。 ②指令种类多。 ③指令寻址方式多。 ④指令功能复杂

C：①CISC 更适于采用硬布线控制逻辑，而 RISC 更适于采用微程序控制。
②CISC 更适于采用微程序控制，而 RISC 更知于采用硬布线控制逻辑。
③ CISC 和 RISC 都只采用微程序控制。
④ CISC 和 RISC 都只采用硬布线控制逻辑。

D：①多寻址方式 ②大容量内存 ③大量的寄存器 ④更宽的数据总线

E：①更简单 ②更复杂 ③不需要 ④不可能

[解析]

本题考查 RISC（精简指令集）计算机有关知识。

在指令系统已确定的情况下，在硬件设计时要尽量使大多数指令能实现"单周期执行"，即在采用流水线组织的计算机中，每个机器周期能完成一条指令功能，而不是一条指令从取指令到完成指定功能只要一个机器周期。另外尽量缩短机器周期，提高主频，从而提高机器运算速度也是硬件设计的主要目标。

采用 RISC 技术的 CPU 硬件一般具有以下特点：

（1） CPU 中寄存器数量相当多；

（2） 采用流水线组织；

（3） 采用硬布线控制逻辑。

RISC 精简了指令系统，使指令种类更少，但各指令所需要的机器周期数相差不大，而且每个机器周期所完成的操作也比较简单，因此没有必要再采用微程序设计。这样，程序经过编译后指令数是增加了，但执行指令的平均机器周期减小了，使得程序的执行时间大大减小。

RISC 指令的简单性编译工作简单化。因为在编译时不必在具有类似效果在许多指令中去进行复杂选择，同时寻址方式少，也不必为选择理想的寻址方式而浪费时间，指令长度固定，指令格式少，这使得优化时更换指令或取消指令变得容易。同时，因为大部分指令能够实现单周期执行，使编译程序比较容易调整指令流。

[答案]A：④ B：① C：② D：③ E：①

试题 3 （1995 年试题 7）
从下列关于 RISC 的叙述中，选出 5 条正确的叙述，并按编码从小到大的次序写在答卷的 A～E 栏内。

①RISC 机器对编译程序的要求比传统的 CISC 低。

② RISC 的 CPU 工艺水平已达到 0.1μm 线宽。

③ RISC 指令招待采用微程序控制方式

④ RISC 的 CPU 采用流水线技术。

⑤ RISC 比传统的 CISC 的 CPU 通用寄存器多。

⑥ RISC 指令格式和寻址方式的种类少。

⑦ RISC 机器用作服务器时性能比传统的 CISC 低。

⑧ RISC 较传统的 CISC 的 CPU 存储器之间的操作功能更强。

⑨ RISC 较传统的 CISC 的 CPU 存储器操作指令内容更丰富，功能更强。

⑩ RISC 只选取一些使用频率高但并不复杂的指令。

[解析]

RISC 即精简指令系统计算机，已在工作站和小型机中广泛采用。

①错误。RISC 机器选取的是使用频率最高的简单指令和很有用但不复杂的指令，从而达到精简指令系统的目的。这种指令系统不及传统 CISC 指令系统丰富，使用起来要求有较高的技巧才能充分发挥 RISC 机器的优点。用户通常用高级语言编程，因而对将高级语言自动翻译成机器指令的编译程序提出了更高的要求，要特别重视编译的优化工作。

②错误。目前超大规模集成电路（VLSI）的工艺水平还达不到 0.1μm 线宽的程度。RISC 的 CPU 作为 VLSI 的一种也不可能要求其工艺水平达到如此细的线宽。它是靠改进其 CPU 内部的系统结构来提高性能的。

③错误。RISC 机器精简了指令系统，每个机器周期所完成的操作也较为简单，因此没有秘要再采用微程序方式来控制指令的执行，且硬布线控制逻辑可提高执行的速度。

④正确。RISC 机器采用流水线技术，使得多条指令的操作可重叠执行，从而使得绝大多数拽令执行的机器周期数仅为 1。

⑤正确。访问存储器要花费较多的时间，RISC 机器为了减少访问次数，通常设有较大的通用寄存器堆，内有许多通用寄存器，并且寄存器间的操作功能也列强，这样许多操作都可在寄存器之间进行。

⑥正确。减少指令格式和寻址方式的种类是指令系统精简的一个重要方面，这样便于发挥流水线的效能，使得许多操作可重叠执行。

⑦错误。实际上，目前用作服务器的大多数工作站和小型机都主要采用了 RISC 的芯片和结构。

⑧正确。见⑤的分析说明。

⑨错误。为了尽可能减少访问内存的次数，大多数 RISC 机器只有很少且非常简单的存储器操作指令。以采用 RISC 技术的 SPARC 处理器芯片为例，它只有两条存储器操作指令，即从内存取数 LOAD 和向存中存数 STORE。

⑩正确。参见对①的分析说明。

[答案]A：④ B：⑤ C：⑥ D：⑧ E：⑩

试题 4 （1993 年试题 11）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

为了大幅度提高处理器的速度，当前处理器中采用了指令及并行处理技术，如超级标量（Superscalar，）它是指 A 。流水线组织是实现指令并行的基本技术，影响流水线连续流动的因素除数数据相关性、转移相关性外，还有 B 和 C ；另外，要发挥流水线的效率，还必须重点改进 D 。在 RISC 设计中，对转移相关性一般采用 E 方法解决。

供选择答案

A：①并行执行的多种处理安排在一条指令内
⑤ 一个任务分配给多个处理机并行执行
⑥ 采用多个处理部件多条流水线并行执行。
⑦ 增加流水线级数提高并行度

B、 C：①功能部件冲突 ②内存与 CPU 速度不匹配 ③中断系统 ④访内指令

D：①操作系统 ②指令系统 ③编译系统 ④高级语言

E：①猜测法 ②延迟转移 ③指令预取 ④刷新流水线重填

[解析]

为了大幅度提高处理器的速度，当前处理器中都采用了指令级并行处理技术，如超级标题就采用了用多个处理部件多条流水线来并行执行指令，在超级标量处理机中配置了多个处理部件和指令译码，采取了多条流水线，还有多个寄存器端口和总线，可以同时执行多悠个操作，以并行处理来提高计算机的速度。

流水线是实现指令并行的基本技术，其基本思想是将一条指令的全过程分为若干段，如取指、译码、取操作数、运算存结果等，每段由不同的功能部件来执行。当流水线各段工作都饱满时，才能达到最高效率。

计算机中不同指令的执行过程并不是规整统一的，有可能不同的指令在不同的操作中用到同一功能部件，使得流水线吵同指令在同一时间访问同一功能部件，这就是功能部个冲突。此时，必然有指令要停下来，从而影响了流水线的效率。

数据有关，即不同甘共苦的指令要访问同一存储单元的数据，如果下一条指令的操作数地址是上一条指令保存结果的地址，那么取操作数的操作就要在保存结果的操作执行完之后才能进行。在遇到条件转移指令时，当条件转移指令进入流水线后，直到下一地址确定之前，流水线不能继续工作而处于等待状态，这也要影响流水线效率。

中断系统工作也和转移情况类似，当中断发生时，要中止当前程序而转入中断程序，此时流水线也会中断。因此数据相关性、转移相关性、功能部件冲突和中断系统都是影响流水线连续流动的因素。好的编译系统产生的目标指令流可使得流水线尽可能满负荷工作。所以，要充分发挥流水线效率，重点是改进编译系统。

在精简指令系统计算机（RISC）中，若遇到成功的条件转移指令或无条件转移指令，流水线使预取的反映指令的失效，从而浪费了机器周期，影响了流水线的效率。为了提高流水线工作效率，RISC 一般将转移指令与其前面的一条指令对换位置，使成功的转移总是在紧跟的指令被执行之后发生，从而使预取的指令不作废，这就是延迟转移法。

[答案]A：③ B：① C：③ D：③ E：②

试题 5 （1991 年试题 11）

从供选择的答案中选出应填入下面叙述中的{ }内的正确答案,把编号写在答卷的对应栏内.

减少指令执行周期数是 RISC 计算机性能提高的基础,它是通过 A 、指令控制部件 B 佩代码、C 寄存器、D 寻址方式和限制访问内存来实现的。E 都是与 RISC 芯片有关的名字。

供选择的答案

A：①超长指令字（VLIW） ②单指令多功能 ③精简指令系统 ④适当扩充指令系统

B：①尽可能多用原有 ②增加 ③软件固化 ④不用

C：①增加 ②减少 ③精选 ④不用

D：①增加 ②减少 ③精选 ④不用

E：①SPARC 和 MIPS ②SPARC 和 SCSI
③MIPS 和 FLOPS ④MIMD 和 EISA

[解析]

精简指令系统计算机 RISC（Reducde Instruction Set Computer）通过精简指令系统，拽令控制部件不用微代码，增加寄存器，减少寻址方式和限制访问内存等方法来减少指令执行周期数，从而提高计算机的性能。因此，减少指令执行周期数是 RISC 计算机性能提高的基础。

SPARC 是 Sun 公司生产的 RISC 芯片的名称，MIPS 是生产 RISC 芯片 R3000 的公司的名称，这两种芯片都是著名的 RISC 芯片。因此，SPARC 和 MIPS 都是与 RISC 芯片有关的名字。

SCSI 是一种小型计算机标准接口，通常用来连接外存储器。FLOPS 表示每秒钟浮点运算的次数。MIMD 为多指令流多数据流，是一种按弗林分类法分类的计算机系统类型。EISA 是一种扩充的工业标准总线系统结构，它们都和 RISC 芯片无关。

[答案]A：③ B：④ C：① D：③ E：①

3.5 综合性试题

最后解析几道综合性试题，涉及到操作系统、网络、并行处理、性能评价等知识点。

试题 1 （1998 年试题 11）

从供选择的答案中选出分别与下述概念最密切相关的术语组，把相应的编号写在答卷的对应栏内。

1．电子商务 A

2.人机界面 B

3．计算机性能评价 C

4.并行处理 D

5．网络体系结构 E

供选择的答案

A～E：

①OSI、对等层协议、无连接服务

②OSI、基准测试程序、TPC

③OCR、基准测试程序、TPC

④吉普森方法、基准测试程序、TPC

⑤EDI、网上商店、身份验证

⑥向量计算机、MPP、多指令流

⑦EDI、网上商店、OCR

⑧所见即所得、语言识别、OCR

⑨EDI、MPP、多指令流

⑩语音识别、OCR、OSI

[解析]

可依次判断供选择的术语组，看哪些与给定的概念相关。

·OSI：Open System Interconnection,开放系统互连，与网络体系结构相关。

·对等层协议：与网络体系结构相关。

·无连接服务：与网络体系结构相关。

·TPC：衡量流水线处理机的性能，与计算机性能评价有关。

·OCR：光学字符识别技术，与人机界面有关。

·基准测试程序：目前比较好的测试计算机性能方法，与计算机性有评价有关。

·吉普森方法：与计算机性能评价有关。

·EDI：Electronic Data Interchange,电子数交换，无纸贸易，与电子商务有关。

·网上商店：以 Internet 为基础，所有卖买包括洽淡、订货、在线付（收款）、开据电子发票都能过 Internet 完成，与电子商务有关。

·身份验证：电子商务中用于验证网络客户的真实身份。

·向量计算机:与并行处理有关.

• MPP：位平面阵列处理机，与并行处理有关。

• 多指令流：同一时刻可执行多条指令，与并行处理有关。

• 语音识别：用软件把人的声音识别为正确的命令或文字，与人机界有关。

[答案]A：⑤ B：⑧ C：④ D：⑥ E：①

试题 2 （1994 年试题 9）

在下列 12 组中，每组中有 3 个词。从该 12 组中，选出组内 3 个词都是与提高计算机硬件处理有关的 5 组,将编号依次写在答案的 A～E 栏内。

①数据库 双机系统 存储器交叉存取

②高速缓冲存储器 虚拟计算机 并行处理

③存储器交叉存取 并行处理 先行控制

④流水线 高速缓冲存储器 存储器交叉存取

⑤双机系统 流水线 结构化程序设计

⑥高速缓冲存储器 存储器交叉存取 先行控制

⑦数据库 批处理 流水线

⑧双机系统 并行处理 虚拟计算机

⑨先行控制 并行处理 流水线

⑩流水线 并行处理 高速缓冲存储器

（11）双机系统 流水线 结构化程序设计

（12）虚拟计算机 批处理 存储器交叉存取

【解析】

根据题意，重点分析与提高计算机硬件处理速度有关的术语和词句。

• 数据库便于大量数据的存取操作和检索，与提高计算机硬件处理速度无关。

• 双机系统能提高可靠性，与提高计算机硬件处理速度无关。

• 存储器交叉存取可提高存储器的存取速度。

·高速缓冲存储器减少了对下一级速度较慢存储器的访问，可提高访问下一级存储器的速度。

·虚拟计算机与提高计算机硬件处理速度无关，虚拟存储器只能扩大程序使用空间，也与提高计算机硬件处理速度无关。

·并行处理可提高硬件处理速度。

·先行控制可预取指令，提高了中央处理器的处理速度。

·流水线使得不同指令的不同操作步骤可在宏观上并行工作，提高了处理速度。

·结构化程序设计是一种较好的模块化程序设计方法，可使程序清晰、易于理解和维护，但与提高计算机硬件处理速度无关。

·批处理是操作系统中一种作业调度的方法，可提高计算机硬件的工作利用率，与提高计算机硬件本身的处理速度无关。

只有③、④、③、③和@组内的 3 个词都是与提高计算机硬件处理速度有关的。

【答案】A：③B：④C：③D：③E：③

试题 3 （1992 年试题 9）
从下列叙述中选出 5 条正确的叙述，把编号依次写在答卷的 A～E 栏内。

①存储器的多体交叉是指将连续地址的存储单元交叉分配到多个模块中，使连续地址的读出高速化。

②操作系统的固化是指用微程序实现指令系统中的复杂指令。

③容错计算机主要用于批处理。

④向量处理机是采用流水线技术通过时间重叠来提高向量运算速度的处理机。

⑤VLIW 计算机是一种新型的体系结构，能充分开发细粒度的并行性，指令长度特别长。

⑥激光打印机是一种印字质量高的高速击打式打印机。

⑦中断屏蔽位为程序员提供了控制中断处理顺序的手段。

⑧在浮点数的加减运算中，首先要进行对阶，即将大阶数的阶码化为与小阶数的阶码相等。

⑨故障诊断就是对机器单件进行测试并确定是否存在故障的过程。

⑩并行性包括同时性（Simultaneity）和并发性（Concurrency）双重含义。

【解析】

①正确。多体交叉存储器将连续地址的存储单元依次分配到多个存储体模块中，每次读写时，多个存储体可同时动作，将多个连续地址的存储单元同时读出。

②错误。操作系统固化是指将操作系统，特别是常用的核心部分用固件来实现，用微程序实现指令系统中的复杂指令并不等于操作系统的固化。

③错误。容错是采用冗余的方法来消除故障的影响，使计算机在出现某些故障的情况下仍可继续运行或降级运行。批处理是操作系统调度作业的一种方法，在批处理系统中，各项工作有序地排在一起形成一个作业流，用户和计算机间不再直接交互信息，由计算机系统自动地、顺序地执行作业流。

容错计算机和批处理之间不存在必然的联系，容错计算机可用于批处理系统，也可用于分时系统或实时系统，特别是实时系统由于要求有更高的可靠性，对容错也有更高的要求。

④正确。向量处理机，有时亦称为向量计算机，例如 Cray－1 等，都是采用流水线技术通过时间重叠的方法来提高向量运算速度的。

⑤正确。超长指令字（VLIW，Very Long InstrUction Word）计算机因其指令宇特别长，有的可达数百位而得名。VLIW 计算机通常具有多个异构的处理单元。一条超长指令可分为若干字段，分别控制各个处理单元和其他功能部件，执行不同的操作，能充分开发细粒度的并行性特性。

⑥错误。击打式打印机指的是靠机械冲击力通过色带等打印出字符或图形的打印机；激光打印机则是利用电子照相原理，即激光放电形成静电影像，吸附墨粉，再转印到普通纸上，最后加热定影印出字符或图形，虽然其印字质量高，但不是击打式打印机。

⑦正确。中断屏蔽位通常在程序状态字中，由程序员编程来置位。当中断屏蔽位被置成“1”（或者等价地，在某些计算机中，中断允许位被置成“0”）时，即使相应的中断源存在，也不允许其提出中断申请或不响应其中断申请。通过这种手段，程序员可以控制中断处理的顺序。

⑧错误。在浮点加减法运算中首先要进行对阶，使参加运算的两个操作数具有相同的阶码，然后只需对其尾数进行相应的运算，对阶的过程总是通过尾数算术右移，将小阶数的阶码增加，以向大阶数的阶码靠拢，而不是将大阶数的阶码向小阶数的阶码靠拢。

⑨错误。对机器硬件测试并确定其是否存在故障的过程称为故障检测。判定故障发生在某个子系统、功能块直至器件的过程称为故障定位。故障诊断应包括故障检测和故障定位两个方面。

⑩正确。并行性包括同时性和并发性双重含义，前者指两个或多个事件在同一时刻发生，后者指两个或多个事件在同一时间间隔内发生，前者比后者要求更为严格。

【答案】A：① B：④ C：⑤ D：① E：(10)

第四章 网络基础知识试题精解

对高级程序员这个层次的人来说，网络知识显得重要多了，最近几年每年至少有一道专门考查网络知识的试题。网络基础知识一个突出的特点是涉及面广、基本概念和术语多，复习时应注意从面上把握一些重要的概念，同时了解一些最新的网络技术知识。

4.1 主要知识点

掌握计算机网络的功能、分类与组成，网络结构与通信，网络协议模型 OSWM 和 TCP / IP 协议以及 Client / Server 结构，了解 Internet / Intranet、网络安全和网络管理方面的知识。

4．1.1 网络的功能、分类和组成

4．1.2 网络的功能

计算机网络具备 3 大基本功能：①通信交往，即计算机之间和计算机用户之间的相互通信与交往；②共享资源，包括硬件资源、软件资源、数据与信息资源；③计算机之间或计算机用户之间的协同工作。

计算机网络的主要用途：①可产生一个性能 / 价格比更好的系统；②提供具有更好可用性和可靠性的应用环境；③在计算机网络内可以通过合理调度实现计算机之间工作负荷的均衡分配；④由计算机网络所构成的系统可以更方便地进行资源扩充和升级换代；⑤可提供友好方便的用户使用界面和计算机资源的有效管理手段。

4．1．1．2 网络的分类

一般按地理区域范围可分为局域网（LAN）、城域网（MAN）、广域网（WAN）、互联网（Int。et）4 种类型。

局域网的基本特征是：

①网络的所有物理设备分布在半径不超过几公里的有限地理范围之内。

②整个网络由同一个组织或机构所拥有。

③在局域网中可实现相当高的数据传输速率，如传输速率范围达 1000MbiUS。

④网络连接相当规整，有严格的标准可遵循。

城域网指所有物理设备分布在同一城区内，覆盖范围约 10km。

广域网的基本特征是：

①网络中信息的传输距离相对很长，可达几千公里以上，涉及到对远程（非本地）计算机的存取与访问。

②通常分属于多个单位或部门所有，资源子网中的各类资源与通信子网分别由各自管辖与负责。

③长距离通信线路上传输速率相对较低，一般在几十 kbit / s 至 ZMbit / S 的数量级之间。

④网络的相互连接结构通常不规整。

互联网实质上是指两个或多个网络互相连接所形成的网络。因特网是全球最大的、开放的、由众多网络相互连接而成的计算机网络。

4．1．1．3 网络的基本组成

计算机网络由计算机硬件、软件、通信设备和通信线路（通信介质）、数据和信息资源所组成。一般把网络分成资源子网和通信子网两大部分。局域网由网络硬件、网络软件、网络 信息资源和应用程序 3 个部分组成。局域网的资源于网由网络中的各台计算机及其外部设备组成；通信子网由传输介质、网卡和网络连接设备组成，比较简单。广域网的通信子网则很
复杂，通常要电信部门负责提供。

4．1．2 网络协议与标准

网络协议（有时也称为通信协议）是指在计算机与计算机之间进行通信必须遵循的一些
事先约定好的规则。网络协议必须遵循标准化的体系结构，目前主要有 ISO 的 OSI 标准和
TCP / IP 协议组标准。

4．1．2 二 ISO 的 OSngM 层次模型

在 OSllRM 层次模型中，把网络协议规定成 7 层模型：

①物理层给出了在一个通信信道的物理媒体上传输原始的二进位数据流时的协议。

②数据链路层给出了把二进位数据流划分成为数据帧，并依照一定规则传送与处理的协议。

③网络层把数据帧划分成更小的“分组”，规定分组的格式，给出使分组经过通信于网正确地从源传送到目的地的协议。网络层提供数据报服务和虚电路服务两类典型的数据分组传送服务方式。

④运输层也称为传输层，根据高层用户的请求建立起有效的网络通信连接。因特网广泛采用的 TCP 协议是一个传输层协议，用来在一个不可靠的互联网上提供可靠的端到端字节流服务协议。

⑤会话层的作用是允许在不同主机上的各种进程之间进行会话。

⑥表示层为应用层提供所传输的信息在表示方面的规则与协议。

⑦应用层为各类不同的网络应用提供使用网络环境的手段，具体规定了用户级别需要
的、带有通信任务的信息服务的规则和协议。

4．1．2．2 TCPffP 协议组

TCPllP 协议组，也称因特网协议组，共有 5 层结构。

物理层和数据链路层与 OSllR 的规定相同，主要实现网络数据的物理存取。

互联网层（即网络层）用来方便地连接各种通信子网，负责选择合适的通信节点，使数据能从源主机发往目的地主机。在互联网层有 IP 协议、ICMP 协议、ARP 协议与 RARP 协议。IP 协议是互联网层的核心协议，负责 IP 分组的传送，ICMP(网际控制报文协议)、ARP(地址转换协议)RARP(反向地址转换协议)等协议则提供辅助的控制功能。

传输层的协议提供主机之间进程与进程的有效数据传输。在传输层中有 TCP 协议与 UDP 协议 TCP 协议即传输控制协议，提供了可靠的面向连接的数据流传输服务规则和约定。用户数据报协 UDP 提供了无连接数据报服务的规则。UDP 协议比 TCP 协议有更好的性能和更高的率，它也使用两个端口来标识源与目的地机器内的端点。

应用层提供了各种应用程序使用的协议，因特网上应用广泛的 HTTP、FTP、Telnet 等服务都是 TCP/IP 应用层协议。应用层对应于 ISO/RM 的会话层、表示层、应用层。

4.1.3 局域网技术

4.1.3.1 主要网络专用设备

主要有网卡、集线器(Hub)、重发器、网桥和交换机等设备。网卡及其驱动程序事实上已基本实现了网络协议中低两层的功能。Hub 是物理层协议级的互联设备，它将多个站点互联起来，也允许将多个网段连到同一个 Hub。重发器是一种在物理层上互联网段的小设备，它放大、增强信号并进行转发以保证信号的可靠传输，重发器连接的两个网段，必须是同一种类型的 LAN。网桥也称为桥接器，是一种在数据链路层把网段互相连接起来的设备。在网桥中可以进行两个网段之间的数据链路层的协议转换。交换机也称为交换器，是在 LAN 中互连多个网段，并可进行数据链路层和物理层协议转换的网骆互联设备。

4.1.3.2 主要传输媒体

目前在网络中数据与信息传输的(有线信道)线路主要采用 3 大类传输媒体，即同轴电缆、双绞线与光纤。双绞线是使铜质导线相互之间按一定扭矩绞合在一起的一种传输媒体，按其是否外加屏蔽层而区分为屏蔽双绞线(STP)和非屏蔽双绞线(UTP)。双绞线也可按其电气特性加以分类，网络中最常用的是 3 类线和 5 类线，目前已有 6 类以上的线。同轴电缆在 LAN 中主要采用幼缆(适用于距离短、站点少的情况)和粗缆(适用于较长距离或较多站点的情况)。光纤由单根玻璃纤维纤心、纤心的包层和塑料保护涂层的组成。根据光在光纤中的传播方式，当前有两大类常用的光纤：单模光纤和多模光纤。

4.1.3.3 LAN 的协议标准和主要的媒体访问控制方式

在 LAN 标准中主要关心物理与数据链路层的网络标准。数据链路层又分为媒体访问控制子层 MAC(与不同的媒体有关)和逻辑链路子层 LLC(与媒体及其访问方式无关)。目前 LAN 有十几个协议标准，统称为 ISO 8802 标准(或 IEEE 802 标准)。

当前使用最广泛的 LAN 是以太网。以太网(Ethernet)也称为 CSMA/CD 总线网。其主要特征是所有 LAN 中的站点本质上都连接在一条公共的总线上，以广播方式通信，采用带有冲突检测(CD)的载波侦听多路访问(CSMA)的媒体访问控制原理进行工作。

令牌环网是由一组连接成环形的计算机站点组成的 LAN，在各个站点之间依照在环上的次序传输令牌的帧信息。常规的令牌环网最高可用 16Mbit/s 工作。与以太网相比较，令牌环网的优点是能提供良好的实时性能和定时保证，因此令牌网主要用于对实时性与可靠性要求更高的一些应用中(如工业自动化控制系统等)。

除了以太网和令牌环网外，目前常用的 LAN 构建方式还有快速以太网、FDDI 光纤网和 ATM 交换式 LAN、千兆位以太网。

4.1.4 广域网技术

4.1.4.1 数据通信

计算机数据通信指的是通过通信信道在计算机之间进行数据与信息的收集、传输、交换或重新分布的一个过程。数据通信涉及到以数字形式或者模拟形式发送数据与接收数据这两个主要环节。通常把发送出数据的来源称为信源或数据源，而把接收数据的目的称为信宿或数据宿。

在数据通信基本模型中把作为信源与信宿的计算机(或其他数字装置)称为DTE(数据终端设备)，把变换器(如调制器)和反变换器(如解调器)称为DCE(数据通信设备)。在DTE与DCE之间应当有标准化的接口，比如MODEM与PC间的接口常用COM串行通信(EIA-RS232C)标准接口，就是物理层协议的一个例子。

数据交换的基本方式有3种：电路交换、报文交换和分组交换。这几种方式有时也可能组合起来使用。

4.1.4.2 网络接入技术

(1)面向家庭、小型商务或小规模应用的主要接入技术。

①普通电话公用网的接入。

②ISDN接入，即窄带的综合业务数字网服务。

③ADSL接入，即非同步数字用户环路接入技术。

④Cable Modem接入，即采用电缆调制解调器在有线电视电缆上进行数据调制。

⑤低轨道卫星网接入，这是目前主要的无线接入技术之一。

(2)面向大型单位与组织机构ISP(指通信子网)的接入技术。

①X.25公用分组交换网接入。

②帧中继接入采用租用专线方式，是一种高速流水线方式的分组交换技术。

③光纤接入，光纤接入技术可分为光纤环路技术(FITL)和光纤同轴混合技术(HFC)。其中光纤环路技术采用全数字传输方式，可细分为光纤到路边(FTTC)、光纤到大楼(FTTB)和光纤到户(FTTH)等。光纤接入可用于各类高带宽、高质量的应用环境中，比如对于大客户，可以以DDN专线、将来的B-ISDN或ATM服务专线等方式接入。

4.1.4.3 路由器和网关

路由器(Router)是实现网络层互连的设备，在OSI标准中称为中间系统。这是一种应用相当广泛的提供网络服务的设备。通常路由器提供了各种速率的多种链路或子网的接口，是一种价格较为昂贵的主动的智能网络节点，一般能参与网络的

管理，提供对资源的动态控制，支持网络工程的实现和协助网络维护活动。路由器的本质特征是：提供网络层的互连，具有路由选功能与流量控制能力。

网关(Gateway)也称为信关，实质上是通信服务器。这是一种相当复杂的在应用层进行网络互连的设备，可以用来连接异种网络(包括不同的各类网络操作系统)，实现网络之间协议转换的功能，故有时也称为协议转换器。

4.1.5 网络的安全性

一个计算机网络良好的安全性由保密性、完整性、可用性、真实性、实用性和占有性等 6 个指标组成。网络的安全性增加了信息安全与路由安全等方面的许多问题。ISO 7498-2 网络安全体系确定了多种基本安全服务和多种安全机制，分别在 OSI 的 7 层中实现网络的安全服务。网络的常用安全机制有标识与验证机制、网络访问控制机制、加密机制、信息完整性机制、认证和审计机制。

4.1.5.1 网络的信息安全技术

(1)访问控制与目录管理

资源的合法使用主要依靠网络管理员进行的必要的访问控制与目录管理，以保证网络资源不被非法用户使用或破坏，也保证数据资源不被非法读出或改写。

(2)数据加密

数据加密算法与技术有许多种。最有名的是 IBM 公司提出的 DES(数据加密标准)加密算法，这是一种对称加密的算法。RSA 算法是比较有名的一类非对称加密算法。所谓"非对称"是指把密钥分开，一个公开密钥用于加密，另一个专用密钥用于解密，这样便于密钥的管理。目前在 Internet 上提供了另一种 PGP 加密系统，这种系统把加密技术交给了公众使用。

(3)身份验证与鉴别

网络的信息安全技术的使用不仅对用户身份进行验证与鉴别，也可以对信息的真实可靠性进行验证与鉴别，用来解决冒充、抵赖、伪造或篡改等问题。除了标识用户加上口令的机制外，最常用的是"数字签名"技术。数字签名可以采用秘密密钥或公开密钥技术实现。签名算法可用 RSA、DSS 数字签名标准等。

(4)两个 TCP/IP 安全协议

Kerberos 和 SSL/SHTTP 都是完整的安全协议，作为保证通信信息安全的协议，应提供下列功能：双方确认认证、数据源的不可否认性、数据接收方的不可否认性、信息完整性保护、以密码保密信息内容以及防重传性(指一个信息包被获取后不能再被重新传输)。

4.1.5.2 防火墙技术

实现防火墙技术的新产品主要有两大类：一类是网络级防火墙，另一类是应用级防火墙。目前一种趋势是把这两种技术结合起来。

(1)网络级防火墙，也称为包过滤型防火墙。事实上这是一种具有特殊功能的路由器，采用报文动态过滤技术，能够动态地检查流过的 TCP/IP 报文或分组头，根据企业所定义的规则，决定禁止或者允许某些报文通过。允许通过的报文将按照路由表设定的路径进行信息转发，相应的防火墙软件工作在传输层与网络层中。

(2)应用级防火墙，也称应用网关型防火墙。大多采用代理服务机制，即采用一个网关来管理应用服务，在其上安装对应于每种服务的特殊代码(代理服务程序)，在此网关上控制与监督各类应用层服务的网络连接。常用的应用级防火墙大致有 3 种类型，分别适合于不同规模的企业内部网：双穴主机网关、屏蔽主机网关和屏蔽子网网关。它们的共同点是需要有一台主机(称为堡垒主机)来负责通信登记、信息转发和控制服务提供等任务。

4.1.6 Internet/Intranet

4.1.6.1 Internet

Internet 目前最主要的服务方式是采用浏览器以 Web 方式入网，以获得大部分的服务项目。主要提供的服务有：远程登录(Telnet)、文件传输(FTP)、电子邮件(E-mail)、网络新闻(News)、查找文件(Archie)、查找人员、以菜单方式浏览信息(Gopher)、按内容自动查找(WAIS)和全球范围的超媒体信息浏览服务(WWW)。

Internet 采用的是 TCP/IP 体系结构网络协议，加入 Internet 的任何网络必须支持 TCP/IP 协议，即在互联网层采用 IP 协议，在传输层采用 TCP(或 UDP)协议。因特网的应用层协议发展较快，目前拥有 Telnet、FTP、SMTP(简单邮件传输协议)、DNS(域名服务协议)、NSP(名字服务协议)和 HTTP(超文本传输协议)等一大批与 Internet 应用服务相关的协议，还有网络管理(如 SNMP)和网络安全等方面的协议。

IP 地址和域名是 Internet 的重要概念。连接到 Internet 上的每台计算机都必须有一个唯一地址。Internet 的地址是用数字来表示的，称为 IP 地址。IP 地址又分为 5 类：A 类、B 类、C 类、D 类和 E 类。大量使用的仅为 A、B、C 这 3 类。A 类地址中第 1 个字节表示网络地址，最高位必须是"0"，而后 3 个字节表示该网内计算机的地址；B 类地址中前两个字节表示网络地址，最高位必须是"10"，后 2 个字节表示网内计算机的地址；C 类地址则是前 3 个字节表示网络地址，最高位必须是"110"，后 1 个字节表示网内计算机地址。A 类地址用于非常巨大的计算机网络，B 类地址次之，C 类地址则用于小型网络。

接入 Internet 的某台计算机要和另一台计算机通信就必须确切地知道其 IP 地址。为便于记忆，采用域名来识别 Internet 上的计算机。

4.1.6.2 Internet

现代的企业网络将以 Internet 作为基础。Internet 的中文名称为内联网。它是把 Internet 技术应用于企业局域网的产物，也是汲取了 Internet 和企业局域网两者的长处而有机地组成的新型企业网络。

Internet 技术的主要特征表现在：采用 TCP/IP 通信协议作为企业网络实现网络通信的基础；采用 Web 技术和 HTTP、SMTP、FTP、SNMP、DNS 等一系列公开协议标准作为企业网络构建时的基本技术；主要着眼于满足企业内部的使用，有控制地向外发布主页信息以及向企业外部提供有限的信息服务；在企业 LAN 的基础上增扩了 Web Server、Mail Server、FTP Server 和 DNS Server 等主要软硬件设施，构成完整的 Internet 网络；采用了有效的网络安全设施和网络管理平台，以保证 Internet 的运行效率和企业的根本利益。

4.1.6.3 Extranet

Extranet 的中文名称为外联网，是企业外部网的意思，是 Internet 概念的进一步延伸与扩展，即把 Internet 构建的技术应用到到了企业与企业之间的网络互联网上，成为企业之间合作的纽带或桥梁。Extranet 把采用 Web 技术构建的企业信息系统的适用范围进一步扩大到了一些特定的外部企业。

构建 Extranet 所采用的技术与 Internet 基本一样，即以 Web 技术、通信互联技术和数据库技术为核心。但是需要考虑外部企业与外部用户的访问需求及相关的安全保证。

4.1.7 客户机/服务器模式

客户机/服务器(Clien/Server)模式，简称 C/S 模式，是网络应用的重要方式。新的网络操作系统和多用户数据库系统基本上都支持 C/S 模式。

两层模式。从数据库管理系统的应用来看，在 LAN 上采用的 C/S 模式，至少拥有一台数据库服务器(DBMS Server)，为各台工作站存取公共数据提供后援支持。把应用任务中的程序执行内容划分成两部分：与数据库存取有关的部分由数据库服务器承担，与应用的人机界面处理、输入/输出或一部分应用的逻辑功能等有关的内容由客户端、工作站承担。

3 层方方式的 C/S 模式，即客户机--功能服务器--数据库服务器。把应用系统的软件相应地分为 3 层：客户机实体内驻留用户界面层软件，负责用户与应用之间进行对话的任务；功能服务器实体内存放有业务逻辑层软件，用来响应客户机的请求，完成相应的业务处理或复杂计算任务；数据库服务器实体内驻留有数据库服务层软件，用来执行功能层发送来的数据库操作，任务完成后逐层地返回给客户机上的用户。

4.1.8 网络计算与电子商务

网络计算是一种高效率和低成本的计算模型，这种模型适合于充分利用Internet 的功能，提供了一类一体化的解决方案。网络计算中所依托的基本技术，除了原有的 TCP/IP 网络协议体系结构与数据库技术外，目前主要有 Web 技术、HTTP 协议标准、CGI 公共网关接口、HTML、XML、Java 应用、瘦客户机技术和 Browser/Server 结构模式。

基于 B/S 模式的信息系统通常采用 3 层或更多层的结构：浏览器--Web 服务器--数据库服务器。以 Web 服务器作为系统的核心，用户端通过 Browser 向 Web 服务器提出查询请求(HTTP 协议方式)，Web 服务器根据需要再和数据库服务器发出数据请求。数据库服务器则根据检索与查询条件返回相应的数据结果给 Web 服务器，最终 Web 把结果翻译成 HTML 或各类脚本(Scripts)语言的相应格式发回至 Browser，用户通过 Browser 浏览所需结果。

电子商务(简称 eB)通常指的是商务贸易活动中各个环节的电子化，其内容可覆盖与商务活动有关的所有方面。eB 是一种典型的网络计算模式的应用，大体上可分为 3 大类，即政府贸易管理的电子化、企业与企业之间的电子商务、企业与个人消费者之间的交易(电子购物)。从技术角度来看，eB 将以Internet/Intranet/Extranet 建设为基础，采用网络计算模式开展商务活动，网络传输与信息的安全性技术有着关键性的重要作用。

4.1.9 网络管理基本概念

网络管理有 5 大功能，即配置管理、失效与故障管理、性能管理、计费管理和安全管理。

TCP/IP 协议组的 SNMP(简单网络管理协议)是网络管理协议的工业标准。目前的网络管理产品大多都支持 SNMP。

企业网主要采用网络管理平台和在该平台上开发出的相应网络管理应用软件进行全企业网范围内的有效网络管理，这些平台与软件都支持开放式的 SNMP 网络管理协议标准。

网络管理平台是实现开放式集成化的网络集中统一管理的一项主要技术，提供了一种软件开发支持环境，在此环境中允许实现网络管理功能的集成。在企业网中，网络管理平台一般采用具有一定性能与规模的通用性网络管理工作站主机，其主要任务包括：搜索与查找网络拓扑，构建、显示与维护企业网的网络拓扑图，构建有关信息的中心数据库，保护、维护与更新该数据库的内容，对网络中出现的各类事件进行监视、记录、分析、处理和过滤，提供友善的网络管理界面和使用方式等。

企业网网络管理平台的代表产品有 IBM 公司的 N、etview fo AIX 或 TME10、CA 公司的 Unicentre TNG|HP 公司的 Open View、Sun 公司的 Net Manager 和 Cabletron 公司的 Spectrum 等。网络设备的主要供应厂商也提供相应的网络管理软件，比如 Cisco 公司的 Works，3Com 公司的 Transcend/Ondemand，Bay Networks 公司的 Qptivity。

4.2 试题分析

网络基础试题主要考查基本知识，难度不大，但涉及面广。从历年试题统计(见表 4-1)来看，自 1994 年以来，每年都有专门考查网络基础知识的试题，主要考查各类网络协议和标准，其中许多试题都涉及到 OSI(开放网络互连)参考模型，与 Internet 有关的的内容也是反复考查的知识点，可以说，Internet 技术、网络协议和标准是考查重点。至今没有网络安全性、客户机/服务器结构、网络计算方面的试题，希望能引起注意。

表 4-1 历年高级程序员级网络基础试题统计

试题	考查知识点
1990 年试题 10	串行通信标准
1992 年试题 7	微机局域网操作系统
1994 年试题 10	以太网
1995 年试题 5	协议概念
1995 年试题 9	OSI 模型
1996 年试题 4	Internet 常识
1997 年试题 3	网络管理
1997 年试题 11	TCP/IP 协议与 Internet
1998 年试题 10	涉及电子商务和网络体系结构
1998 年试题 11	网络互连设备与 OSI 模型
1999 年试题 13	宽带网络接入技术
2000 年试题 13	网络协议，涉及网络管理、移动上网、拨号上网、路由选择

试题 1 (2000 年试题 13)
从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。如：

A：用于网络管理；

B：用于移动电话无线上网；

C：可用于家庭拨号上网；

D：一种面向比特的数据链路通信规程；

E：一种路由选择协议。

供选择的答案

A～C：①SNMP ②PPP ③RIP ④WAP

D、E：①OSPF ②HTTP ③HDLC ④RARP

【解析】

本题考查常用的网络协议知识，涉及最新的移动无线上网技术。

这里简单补充一下有关知识。

SNMP 是基于 TCP/IP 的简单网络管理协议。

PPP 是用于拨号接入，实现点对点通信的远程访问协议。

RIP(路由信息协议)是最常用的内部路由协议，多用于 UNIX 系统中。

OSPF(开放式最短路径优先)是一种为 TCP/IP 开发的连接状态协议，也是一种内部路由协议，多用于路由器中。

WAP 是用于移动电话(手机)无线上网的协议。

HTTP 是超文本传输协议。

HDLC 是国际标准化组织(ISO)制定的高层数据链路控制协议。

RARP(反向地址转换协议)用于动态实现 IP 地址向物理地址的转换。

【答案】A：① B：④ C：② D：③ E：①

试题 2 (1999 年试题 13)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

向端用户提供尽可能宽带的网络接入是引起人们广泛关注的技术。A 只能提供 128kbit/s 的接入数据速率， B 则是通过电话双绞线向端用户提供更高信息传输带宽的一种接入技术，而采用 C 和电缆调制解调器(Cable Modem)也可获得和后者同样数量级的接入带宽。第 3 代无线通信的 D 可提供高达 2Mbit/s 的接入数据速率。光纤到户，即 E ，则是将来的一种发展方向。

供选择的答案

A、B：①B-ISDN ②N-ISDN ③CDMA ④ADSL

C、D：①HFC ②GSM ③CDMA ④HDSL

E：①FDDI ②FTTH ③FTTC ④FTTB

【解析】

本题考查网络接入技术知识，涉及的内容比较新、比较多。这里用较大的篇幅详细介绍试题中涉及的若干知识点。

B-ISDN(Broad Integrated Service Digital Network)，中文名称是宽带综合业务数字网。B-ISDN 所涉及的关键技术主要有 ATM 宽带交换机、同步光纤网/同步数字系列和光纤同步网。B-ISDN 以光纤为传输媒介，这一方面保证了所提供的业务质量，同时也减少了网络运行中的诊断、纠错、重发等环节，从而提高了网络的传输速率，B-ISDN 的用户/网络接口的速度可高达 150Mbit/s 或 600Mbit/s。另外，B-ISDN 可以提供多种高质量视频信息，如增强型可视电话、高清晰度电视等。B-ISDN 还可以将电信与广播、电视合为一体，即将广播电视业务也纳入到电信业务之中。

根据不同形式宽带通信及其应用，B-ISDN 有两种主要的业务类别：交互型业务和分配业务。交互型业务又分为 3 种业务：会话型业务、电子信箱业务和检索型业务。分配业务又由两种业务来表示：客户不能进行单独演示控制的分配型业务和客户能够进行单独演示控制的分配型业务。
B-ISDN 向客户提供的业务将有以下特点：①高度的综合性；②图像通信占有重要的地位；③大量使用多媒体业务；④客户从网络得到的业务将是大量的、多种多样的；⑤终端工作速率可变；⑥终端将一机多用。

N-ISDN(Narrow Integrated Service Digital Network)，中文名称是窄带综合业务数字网，又称"一线通。""一线通"采用数字传输和数字交换技术，利用现有的电话网将电话、传真、数据、图像等多种业务综合在一起进行传输和处理，实现了端到端的数字连接，可实现会议电视、远程教学、远程医疗和多点采集。

"一线通"在上网的同时还能利用另一个 B 通道进行电话交流或收发传真。若将两个 B 通道捆绑成一个通道使用，速度将达 128kbit/s，

N-ISDN 主要有基本速率扫口和基群速度接口两种接口。基本速率接口主要面向现有电话网的普通用户、上网用户以及小型单位，提供 2B+D(144kbit/s)的基本速率，其中 B 信道为 64kbit/s，D 信道为 16kbit/s。基群速率接口主要面向企业用户和集团用户，提供会议电视等高速通信业务，速率为 30B+D(2Mbit/s)，其中 B 信道为 64kbit/s，D 信道为 16kbit/s(注：B 信道为传送信息的信道，D 信道为传送信令的信道。)

CDMA(码分多地址移动通信技术)与 FDMA(频分多地址移动通信技术)和 TDMA(时分多地址移动通信技术)相比，CDMA 具有许多独特的优点，其中一部分是扩频通信系统所固有的，另一部分则是由软切换和功率控制等技术带来的。CDMA 移动通信网是由扩频、多址接入、蜂窝组网和频率再用等几种技术结合而成，含有频域、时域和码域三维信号处理的一种协作，因此它具有抗干扰性好，抗多径衰落，安全保密性高，同频率可在多个小区内重复使用，所要求的载干比(C/I)小于 1，容量和质量之间可做权衡取舍等属性。系统容量大，理论上 CDMA 移动网比模拟网大 20 倍。实际要比模拟网大 10 倍，比 GSM 要大 4～5 倍。这些属性使 CDMA 比其他系统有更大的优势。

CDMA 移动通信网的关键技术有：①功率控制技术；②PN 码技术；③RAKE 接收技术；④软切换技术；⑤话音编码技术。

xDSL 是 DSL(Digital Subscriber Line)的统称，意即数字用户线路，是以铜电话线为传输介质的传输技术组合。DSL 技术在传统的电话网络(POTS)的用户环路上支持对称和非对称传称模式，解决了经常发生的在网络服务供应商和最终用户间的"最后 1 公里"的传输瓶颈问题。由于电话用户环路已经大量铺设，如何充分利用现有的铜缆资源，通过铜质双绞线实现高速接入就成为业界的研究重点，因此 DSL 技术很快就得到重视，并在一些国家和地区得到大量应用，其中"x"代表着不同种类的数字用户线路技术。各种数字用户线路技术的不同之处，主要表现在信号的传输速率和距离以及对称和非对称的区别上。DSL 技术主要分为对称和非对称两大类。

第一类：对称 DSL 技术

(1)HDSL--High-bit-rate DSL(高速度 DSL)

HDSL 是 xDSL 技术中最成熟的一种，已经得到了较为广泛的应用。其特点是：①利用两对双绞线传输；②支持 Nx64kbit/s 和种速率，最高可达 E1 速率；③HDSL 是 T1/E1 的一种替代技术，主要用于数字交换机的连接、高带宽视频会议、远程教学、蜂窝电话基站连接、专用网络建立等。
与传统的 T1/E1 技术相比，HDSL 具有以下优点：①价格便宜；②容易安装(T1/E1 要求每隔 0.9～1.8km 就安装一个放大器，而 HDSL 可在 3.6km 的距离上传输而不用放大器。)

(2)SDSL--Symmetric DSL(对称 DSL)

①利用单对双绞线；②支持多种速率到 T1/E1；③用户可根据数据流量，选择最经济合适的速率，最高可达 E1 速率，比用 HDSL 节小一对铜线；④在 0.4mm 双绞线上的传输距离可达 3km 以上。

(3)MVL--Multiple Virtual Line(多虚拟数字用户线)

MVL 是 Paradyne 公司开发的低成本 DSL 传输技术。

①利用一对双绞线；②安装简便，价格低廉；③功耗低，可以进行高密度安装；④用与 ISDN 技术相同的频率段，对同一电缆中的其他号干扰非常小；⑤支持语音传输，在用户端无需语音分离器；⑥支持同一条线路上同时连接多至 8 个 MVL 用户设备，动态分配带宽；⑦上/下行地速率可达 768kbit/s；⑧传输距离可达 6km。

第二类：非对称 DSL 技术

ADSL-Asymmetric DSL(非对称 DSL)能够在现有的铜双绞线，即普通电话线上提供高达 8Mbit/s 的高速下行速率，远高于 ISDN 速率；而上行速率为 1Mbit/s，传输距离达 3km～5km。其优势在于可以充分利用现有的铜缆网络(电话线网络)，在线路两端加装 ADSL 设备，即可为用户提供高宽带服务，由于不需要重新布线，降低了成本，进而减少用户上网的费用。

ADSL 的主要特点：①具有很高的传输速率，ADSL 能在现有普通电话线上以很高的速率传输数据，下行达到 8Mbit/s，上行达到 1Mbit/s；②上因持网和打电话互不干扰；③提供多种先进服务；④独享带宽，ADSL 的每个用户都独享 8Mbit/s 带宽；⑤费用低廉；⑥安装方便快捷。

另外还有 VDSL(Very-high-data-rate DSL)等 DSL 技术，这些技术可在较短的距离上提供极高的传输速率，但应用还不是很多。

所谓光接入网(OAN)就是采用光纤传输技术的接入网，泛指本地交换机或远端模块与用户之间采用光纤通信或部分采用光纤通信的系统。通常，OAN 指采用基带数字传输技术，并以传
输双向交互式业务为目的接入传输系统，将来应能用数字或模拟技术升级传输带宽广播式和交互式业务。

ONU 的作用是为光接入网提供直接的或远端的用户侧接口。按照 ONU 在接入网中所处的位置不同，可以将 OAN 划分为几种基本不同的应用类型，即光纤到路边(FTTC)、光纤到楼(FTTB)以及光纤到家(FTTH)和光纤到办公室(FTTO)。

(1)光纤到路边(FTTC)

在 FTTC 结构中，ONU 设置在路边的入孔或电线杆上的分线盒处，此时从 ONU 到各个用户之间的部分仍为双绞线铜缆。若要传送宽带图像业务，则这一部分可能会需要同轴电缆。

FTTC 结构主要适用于点到点或点多点的树形--分支拓扑结构，用户多为居民住宅用户和小企事业用户，典型用户数在 128 个以下，经济用户数正逐渐降低到 8～32 个乃至 4 个左右。

(2)光纤到楼(FTTB)

FTTB 也可以看作是 FTTC 的一种变形，不同处在于将 ONU 直接放到楼内(通常为居民住宅公寓或小企事业单位办公楼)，再经多对双绞线，将业务分送给各个用户。FTTB 是一种点到多点结构，通常不用于点到点结构。FTTB 的光纤化程度比 FTTC 更进一步，光纤已铺设到楼，因而更适于高密度用户区，也更接近于长远发展目标。预计会获得越来越广泛的应用，特别是应用于那些新建工业区或居民楼以及带宽传输系统共处一地的场合。

(3)光纤到家(FTTH)和光纤到办公室(FTTO)

在原来的 FTTC 结构中，如果将设置在路边的 ONU 换成无源光分路器，然后将 ONU 移到用户家，即为 FTTH 结构。如果将 ONU 放在大企事业用户(公司、大学、研究所、政府机关等)终端设备处，并能提供一定范围的灵活的业务，则构成所谓的光纤到办公室(FTTO)结构。由于大企事业单位所需业务量大，因而 FTTO 结构发展很快。考虑到 FTTO 也是一种纯光纤连接网络，因而可以归入 FTTH 一类的结构。然而，两者的应用场合不同，结构特点也不同。FTTO 主要用于大企事业用户，

业务量需求大，因而在结构上适于采用点到点或环形结构。而 FTTH 用于居民住宅用户，业务量需求大，因而在结构上适于采用点到点或环形结构。而 FTTH 用于居民住宅用户，业务量需求很小，因而经济的结构必须是点到多方式。FTTH 和 FTTO 主要特点可以总结如下：由于整个接入网是全透明光网络，因而对传输制式(例如 PDH 或 SDH，数字或模拟等)、带宽、波稀薄和传输技术没有任何限制，适于引入新业务，是一种最理想的业务透明网络，是接入网发展的长远目标。

HFC(Hybrid Fiber Coax，即光纤同轴混合)是在有线电视网(CATV)的基础上发展起来的一种宽带接入技术，其网络结构为树型或总线型。从前端至小区光节点采用光纤传输，从光节点到用户终端采用同轴电缆，小区光节点所管辖的用户数一般在 500 左右(理想情况)，整个光节点内部网可提供约 1GHZ 的带宽，由光节点内的所有用户共享。在 HFC 前端、语音、数据、视频等多媒体信息经调制后与电视射频信号一起伟输到用户端，经过分离器提取多媒体信息，并恢复成数据包形式,通过类似 IEEE 802.3 的共享介质接入机制 MLAP(MAC Level Access Protocol)进入用户终端设备。此外,前端的带宽控制器可通过编程方式对用户 Cable Modem 进行频率分配控制。

Cable Modem 是 HFC 中的关键设备，按照其上下行信道传输速率是否对称分为两类：一是上下信道对称，双向通信速率最高可达 100Mbit/s，适用于高速 LAN 互联；另一种是上下信道不对称，上行速率最高为 768kbit/s，下行速率最高可达 30Mbit/s，适用于高速 Internet 接入等。
HFC 的特点是除了能提供原有的视频信号外，还能提供双向语音、数据、视频等宽带多媒体信息业务，现阶段具有一定的发展前景，但属于过渡型技术。

【答案】A：② B：④ C：① D：③ E：②

试题 3 (1998 年试题 10)

本题是一道综合题，涉及电子商务和网络体系结构，参见第三章 3.5 节试题 1。

试题 4 (1998 年试题 11)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

有多种设备可以实现不同网段或网络之间的互连，互连设备通常可按工作在 OSI 模型中的层次来划分。在物理层实现互连的称为 A ；在数据链路层实现互连的称为 B ；在网络层实现互连的称为 C ；在运输层及以上高实现互连的设备称为网关或 D 。E 也是一种用来构造局域网的常用设备，通常可以用双绞线把服务器与 PC 客户机等连入 E 。

供选择的答案

A～E：①集线器 ②协议转换器 ③网桥 ④路由器
⑤网关 ⑥转发器

【解析】

网络互连需要通过一个中间设备或中间系统，术语称之为中继系统(Relay System)，根据中继系统在网络中所处的层次，可以有以下几种中继系统：

物理层中继系统，即转发器(Repeater)；

数据链路层中继系统，即网桥或桥接器(Bridge)；

网络层中继系统，即路由器(Router)；

网桥和路由器的混合系统，即桥路器(Brouter)；

网络层以上的中继系统，即称为网关(Gateway)。网关也称为网间连接器、信关或联网机。用网关连接两个不兼容的系统要在高层进行协议转换，因此，网关也称为协议转换器。

双绞线以太网问题问题和集线器(Hub)一块使用的，集线器也是一种用来构造局域网的常用设备，通常可以用双绞线把服务器与 PC 客户机等连入集线器。

【答案】A：⑥ B：③ C：④ D：② E：①

试题 5 (1997 年试题 3)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

当网络用户通过网络与另一台主机 X 通信，发现响应太慢时，可运行 A 程序，把一个分组发向主机 X，通过查看所返回的分组首部的 B ，发现问题的所在，并采取相应的措施。

对于一个大中型网络，需要有一个网络管理系统进行管理。当前流行的各类网管平台软件都支持 C 协议。驻留在 D 上的网管平台软件可通过该协议软件调阅被管的网络结点内的管理信息库中的内容。有若干常用的网络平台软件，但 E 不是网管平台软件。

供选择的答案

A：①Browser ②Bitware ③Ping ④Handshaking

B：①地址 ②时间戳 ③标识码 ④校验码

C：①MAP ②SNMP ③MHS ④FTAM

D：①数据库服务器 ②通信服务器 ③主路由器 ④网络管理工作站

E：①Net Manager ②Open View ③Sun View ④Net View

【解析】

该题主要测试考生对网络基本概念和常用方法的掌握程度。

在使用 TCP/IP 协议的网络中，Ping 是一个常用的网络测试程序，它发送一个 ICMP(Internet Control Message Protocol)的请求应答包给一个主机网关，以期对方能发回一个 ICMP 的应答包。发送方可以通过检查返回包首部的时间戳来获得应答时间的长短信息。因此当网络用户通过网络与另一台主机 X 通信，发现响应太慢时，可运行 Ping 程序，把一个分组发向主机 X，通过查看所返回的分组首部的时间戳，发现问题的所在，并采取相应的措施。

IP 提供的是不可靠非面向连接的数据报服务。为了使网上的计算机能够检测报告差错，或提供有关意外情况的信息，设计者在 IP 协议中提供了一种特殊作用的报文制，即 ICMP。它位于 IP 数据报的数据部分，同 IP 数据报的数据信息一样在网上传输，但其目的地不是目的计算机上的一个用户进程，而该计算机上的 IP 软件。当一个有错误的 ICMP 报文到达时，IP 软件模块就处理本身的问题，不再将其提供给高层。

Browser 是在网络上进行信息浏览的软件--浏览器，目前常用的有 Netscape 和 Internet Explorer。Bitware 是一个利用调制解调器(Modem)收发传真的软件。Handshaking(握手)是网络上进行信息传输时传输双方为达到同步所做的操作的通用说法。

目前常用网络平台软件有 Sun 公司的 Net Manager，HP 公司的 Open View，IBM 公司的 Net View。Sun View 不是网络平台软件，它是 Sun 公司的操作系统 SunOS 上的视窗系统。一个大中型网络，需要有一个网络管理系统进行管理。网络管理最通用的协议是 SNMP(Simple Network Management Protocol)，即简单网络管理协议。当前流行的各类网管平台软件都支持这一协议，它主要用于在网络中不同站点之间交换与网络管理有关的信息，通过网络管理软件可监视和控制网络的配置及运行情况，测试、记录和统计网络中各处的数据流量和差错情况等。当前流行的各类网管平台软件都支持 SNMP 协议。驻留在网络管理工作站上的网管平台软件可通过该协议软件调阅被管的网络结点内的管理信息库中的内容。有若干常用的网络平台软件，全 Sun View 不是网管平台软件。

MHS(Message Handling System)是网络上传送标准电子邮件的报文处理系统。FTAM(File Transfer Access And Management)是指在网络上不同主机间的文件传送访问和管理。MAP(Manufacturing Automation Protocol)是指制造自动化协议，是由美国通用汽车公司提出的一种用于生产自动化的局域网协议，它为众多来自不同厂家的各种设备集成提供一个标准的、开放的通信网络环境。

网络上存在不同功能的服务器：数据库服务器提供对数据库数据的管理，并为数据库应用程序提供数据访问服务；通信服务器一般提供很多 I/O 端口，可以同时为许多客户提供通信服务；网络上的路由器负责不同网络、不同协议之间数据包的转发。这些设备与网络管理平台软件没什么关系。网络管理工作站提供对所负责的网络的监控，所以驻留在网络管理工作站上的网管平台软件可通过该软件调阅被管的网络结点内的管理信息库中的内容。

【答案】A：③ B：② C：② D：④ E：③

试题 6 (1997 年试题 11)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

计算机的发展已进入了网络计算的新时代。Internet 是目前世界范围内最大的互联网。如此多的各种计算机之所以能通过 Internet 相互通信，是因为它们遵循了一套共同的 Internet 协议。这套协议的核心是 A ，在其上建立的无连接的运输层协议是 B ，万维网 WWW 上超文本传输遵循 C ，电子邮件传输遵循 D ，Ethernet 与 Internet 连接时要用到 E 。

A～E：①TCP ②IP ③EIDFACT ④HDLC
⑤ARP ⑥UDP ⑦FTP ⑧ICMP
⑨SMTP ⑩HTTP

【解析】

Internet 是用一种路由器(专用计算机)将网络互联组成的。互联的计算机需要软件才能通信，通信双方必须遵循的规则称为协议。Internet 中使用的一个关键协议是网际协议 IP(Internet Protocol)，IP 详细定义了计算机之间进行通信应遵循的规则，例如定义了分组的格式，以及路由器如何将每一个分组递交到目的地等。

Internet 中的分组称为 IP 数据报。为了使路由选择和数据报的递交成为可能，连接到 Internet 上的每台计算机都必须被指定一个唯一的地址，该地址称为 IP 地址。在 IP 上层建立的传输层有两个并列的协议：ICP 和 UDP，其中 TCP(Transport Control Protocol)是面向连接的，UDP(User Datagram Protocol，用户数据报协议)是无连接的。一般情况下，TCP 和 UDP 并存于 Internet 中，TCP 提供高可靠性服务，其协议非常复杂，执行效率低，UDP 提供高效率服务，其协议相对简单得多。

WWW 是 World Wide Web 的缩写，是把信息检索技术和超文本(Hyper Text)技术相结合而形成的环球信息系统。超文本是一种描述信息的方法，其文本中的被选词可以被扩展，以提供该词所关联的其他信息，即把这些词"链接"到其他文档上，这些文档可能是以文本、图片或其他形式提供的。超文本技术是指在一文档中的某些词暗藏着与其他文档的联系，通过这些词(如用鼠标单击这些词)就能迅速找

到相关文档，并可把它调来阅读或作其他处理。在 WWW 中，与某一文档相关联的文档可以放置在世界各地的其他计算机内。WWW 的成功在于制定了一套标准的容易掌握的超文本标记语言 HTML(Hyper Text Markup Language)、统一资源定位器 URL (Uniform Resource Locator)和超文本传输协议 HTTP(Hyper Text Transport Protocol)。WWW 提供一种高级浏览服务，采用客户机/服务器(Client/Server)模式工作。Internet 上的一些计算机运行着 WWW 服务器程序，它们是信息的提供者，在用户计算机上运行着 WWW 客户程序，帮助用完成信息咨询。超文本传输协议 HTTP 是 WWW 客户机和服务器在网际上响应用户请求并传输信息的协议。当用户激活一个"链接"后，服务器使用 HTTP 通过送回约定好格式的文件来作出响应，客户机通过一个浏览器来显示响应信息。

TCP/IP 协议提供两个电子邮件传输协议：MTP(Mai Transfer Protocol，邮件传输协议)和 SMTP(Simple Mail Transfer Protocol，简单邮件传输协议)。在 Internet 中，大部分电子邮件由 SMTP 发送，其特点就是简单，它只定义邮件如何在传输系统中通过发方和收方之间的 TCP 连接进行传输而不规定其他操作。

在 Internet 中，IP 层以上各层都使用 IP 地址，而 IP 地址和计算机的物理地址是不同的，在物理网络内部，仍使用各自原来的物理地址，这样，在 Internet 中存在两种地址，二者之间要建立映射关系，包括从 IP 地址到物理地址的映射和从物理地址到 IP 地址的映射。Ethernet 网络具有广播能力，与 Internet 连接时，遵循 ARP 协议(Address Resolution Protocol)。ARP 的原理如下：假设在某广播型网络上，主机 A 要得到主机 B 的物理地址，首先主机 A 广播一个 ARP 请求报文，请求 Internet 地址 IB 的主机回答其物理地址 PB。网上所有主机(包括 B 机)都将收到该请求，但只有 B 机能够识别出自己的 IB 地址，并向 A 发回应答，回答 B 机的物理地址 PB，以后就可直接在 A 机和 B 机之间传输数据了。

【答案】A：② B：⑥ C：⑩ D：⑨ E：⑤

试题 7 (1996 年试题 4)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

Internet 提供的服务有通信、远程登录、浏览和检索等。A 直接用于人际通信，B 用于远程登录。 C 不是浏览软件。在浏览软件中， D 不支持 HTML，E 是目前微机上最常使用的浏览器。
供选择的答案

A、B：①WWW ②E-mai ③URL ④TCP/IP
⑤Telnet ⑥Lycos ⑦HTML ⑧PPP

C～E：①Hotjava ②Netscape ③Mosaic ④Lycos ⑤Gopher

【解析】

Internet 即国际互连网络，现一般译为因特网。Internet 是目前世界上最大、用户最多的互联网络。TCP/IP 是传输控制协议/互连网协议，Internet 使用 TCP/IP 协议，把不同的局域网和广域网通过路由器互相连接，形成一个全球性大网。

Internet 提供各种服务职通过信、远程登录、新闻组、电子邮件、文件传输、信息的测览与检索等。电子邮件(E-mail)用网络用户间的通信；FTP 以客户机/服务器方式为用户提供文件舆服务，用户可以从文件服务器上下载自己感兴趣的文件，也可以将自己的文件传送到服务器上；远程登录 Telnet 使得用户可通过运行 Telnet 把自己的计算机作为服务器的一个操作终端，调用服务器的共享资源。

Internet 上的海量信息，使得用户查找自己感兴趣的信息变得费时费力，因而好的浏览器会成为用户上网的好帮手，节约上网时间，提高上网效率。Gopher 是一咱交互式、菜单驱动的 Internet 浏览器。WWW(World Wide Web 译为万维网，它使用超文本舆协议 HTTP(Hyper Text Transport Protocol)向用户提供服务，在 Internet 上有很多 WWW 服务器，提供 WWW 服务。要访问这些服务器，用户可在客户机上运行 WWW 浏览器，该程序可以解释和显示从 WWW 上找到的超文本文档。目前的 WWW 浏览器有 Mosaic、Netscape Navigator、Hotjava 等，Netscape Navigator 是目前微机上较流行的 WWW 浏览器。Hotjav 可运行于 WindowsNT、Windows95 之上，支持用 java 编制的可执行信息，上述浏览软件中，只有 Gopher 不支持 HTML。

HTML(Hyper Text Markup Language)超文本标记语方是一种描述语言，用它编制 WWW 上的超文本文档，支持超文本链接，可以从一个文档指向另一个文档，或是从文档的一部分指向另一部分。PPP(Point-to-Point Protocol)是一个点到点的信息舆协议，是用户计算机通过 Modem 与 Internet 相连的一种方式。
URL(Universal Resource Locator)统一资源定位器，用于确定 Internet 上的地址。Lycos 是 Internet 上一个信息检索工具，不是浏览器。

【试题】A：②B：⑤C：④D：④E：②

试题 8 (1995 年试题 5)

从供选择的答案中，选出应填入下面叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

协议是一组 A ，它有助于 B 之间的相互理解和正确进行通信。协议中有 3 个鲞关键因素。其中 C 定义数据的表示形式；D 则能使数据管理所需的信息得到正确理解，E 则规定了通信应答信号之间的间隔和先后关系。

供选择的答案：

A、B：①软件 ②外部设备 ③通信实体 ④时钟 ⑤约定的规则 ⑥寄存器组 ⑦存储器 ⑧CPU

C、D、E：①媒体 ②语义 ③文本 ④语言 ⑤时序 ⑥编码 ⑦语法⑧波特率⑨文件

【解析】

协议就是两个计算机之间通信时对传送信息内容的理解、信息表示形式以及各种情况下的应答信号都必须遵循的共同约定。在 ARPA 网中将协议按功能分成若干层次，分层标准和各层中采用的具体协议的综合就构成了网络的体系结构。

为了减少协议设计的复杂性，大多数网络都按 OSI 的 7 层结构来组织，每一层都有协议，但不管是何种网络，哪一层的协议，作为协议必须具备 3 个要素，即语法、语义和时序。语法是一种形式化的表示，它定义了传输数据的表示形式；语义则定义了数据的实际含义，从而使数据管理所需的信息得到正确理解；时序则规定了通信实体之间应答信号的相互间隔和顺序关系。

【答案】A：⑤ B：③ C：⑦ D：② E：⑤

试题 9 (1995 年试题 9)

从供选择的答案中，选出应填入下面关于 OSI 叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

国际标准化的开放系统互连(OSI)参考模型共分 7 层。其中，处理系统之间用户信息的语法表达形式问题的是 A 层；规定通信双方相互连接的机械、电气、功能和规程特性的是 B 层；向用户提供各种直接服务，如文件传送、电子邮件、虚拟终端等的是 C 层；通过校验和反馈重发等方法将原始不可靠的物理连接改造成无差错的数据通道的是 D 层；负责通信子网中从源到目标路径选择的是 E 层。

供选择的答案：
A～E 层：①物理层 ②数据链路层 ③网络层 ④运输层
⑤会话层 ⑥表示层 ⑦应用层

【解析】

在国际标准化组织公布的开放系统互连基本参考模型中，详细规定了该模型的 7 个层次，并对每一层在网络传输中所各自负责的具体任务做了详尽的描述。

OSI 模型共分 7 层，自上而下依次为应用层、表示层、会话层、传输层、网络层、数据链路层和物理层。下层实体为上层实体提供服务，上层实体为下层实体提供接口。具体请参见相关的知识点综述。

【答案】A：⑥ B：① C：⑦ D：② E：③

试题 10 (1994 年试题 10)

从供选择的答案中，选出应填入{}内的正确答案，把编号写在答卷的对应栏内。

以太网遵循 IEEE802.3 标准，用粗缆组网时每段不能大于 A 来。超过上述长度时，要分段，段间用 B 相连。同时，整个网的总长度不能大于 C 米。若总长度超过上述长度，则需分成两个网，网之间用 D 相连。这是在 ISO 的 OSI 的模型中 E 层的连接。

供选择的答案

A～C：①50 ②100 ③185 ④500
⑤1000 ⑥2500

B、D：①网络适配 ②重发器 ③调制解调器 ④网桥

E：①物理 ②数据链路 ③网络 ④运输

【解析】

以太网(Ethernet)遵循美国电气电子工程师协会 IEEE 制定的 802.3 标准，目前已成为国际标准化组织 ISO 的国际标准 ISO 802.3，是一种带冲突检测的载波侦听多路访问(CSMA/CD)技术，其网络物理介质可以采用双绞线、细同轴电缆、粗同轴电缆或光纤。10Base5 表示网络是采用同轴电缆的标准构建而成；10Base2 表示细同轴电缆；10BaseT 表示双绞线；FDDI 表示光纤。

受发送器的功率影响以及接收器的分辨能力及媒体上信号传输衰减和畸变的限制，10Base5 标准规定每段同轴电缆的长度不能超过 500m。超过上述长度时，段与段之间要用重发器(Repeater)相连。Repeater 实际上是把从一段中接收到的信号放大后，再向另一段发送出去，有时也译为中继器。

重发器不能无限制地使用。由于以太网冲检测的需要，全网内端到端最大的信号传播延迟时间不能超过一定的限制值，这使得整个网的总长度不能大于 2500m。若总长度超过了 2500m，要分成两个网，网之间再用网桥(Bridge)来相连，这是在 ISO 的开放系统互连(OSI)七层模型中数据链路层的相连，用重发器相连则是在物理层的连接。两个网络在网络层连接通常由路由器(Router)来完成，在运输层以上的高层连接的网际互连设备则称为网关(Gateway)。网络适配器是将终端接入网络的设备，如常见的各种网卡；调制解调器(MODEM)则是将离散的数字信号转换成连续的模拟信号以便在模拟信道上传输的设备。

【答案】A：④ B：② C：⑥ D：④ E：②

试题 11 (1992 年试题 7)

从供选择的答案中选出应填入下面关于网络操作系统叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

微机局域网操作系统能支持网络中的服务器与工作站的通信连网。通常此类操作系统可由 A 、B 和 C 所组成。A 是专门提供网络服务的。设置的网络管理软件，它直接与网络服务器硬盘和网卡 D 连接，对用户访问服务顺的权限和数据加密进行管理，对多个用户访问服务器共享资源进行管理。C 用于实现多个网络的互连服务。此类操作系统中的协议模块与 OSI 协议建立起对应关系，比如： E 对应于会话层。

供选择的答案

A～E：①路由器软件 ②NETBIOS 仿真程序 ③系统管理服务程序
④网络文件服务器程序 ⑤驱动程序 ⑥工作站 Shell 程序

【解析】

微机局域网操作系统能支持网络中的服务器与工作站的通信连网，通常此类操作系统由网络文件服务器程序、工作站 Shell 程序和路由器软件所组成。

网络文件服务器程序是专为提供网络服务设置的网络管理软件，它直接与网络服务器硬盘和网卡驱动程序连接，对用户访问服务器的权限和数据加密进行管理，对多个用户同时访问服务器共享资源进行管理。

工作站 Shell 程序驻留在各结点上，为本地的应用提供各类网络服务，路由器软件用于实现多个网络的互连服务。此类操作系统中的协议模块与 OSI 建立起对应关系，比如，NETBIOS 仿真程序对应于会话层。

【答案】A：④ B：⑥ C：① D：⑤ E：②

试题 12 (1990 年试题 10)

从供选择的答案中选择应填入下面叙述中的{ }内的正确答案，把编号写在答卷的对应栏内。

RS-232-C 是目前常见的一种接口标准，它是由 A 提出制定的。该标准在 OSI 模型中属于 B 层协议标准，通过 RS-232-C 来连接两个设备最少要连接 C 条线。这个标准的设计数据速率是处理 D bit/s。在 Dbit/s 条件下，采用 RS-422 标准来代替 RS-232-C，连接设备间的距离可扩展到约为原有的 E 倍。

供选择的答案

A：①CCITT ②EIA ③IFIP ④IEEE

B：①物理 ②数据链路 ③网络 ④运输 ⑤会话

C、E：①2 ②3 ③4 ④7
⑤10 ⑥25 ⑦80 ⑧100

D：①4800 ②9600 ③19200 ④20000
⑤6400

【解析】

RS-232-C 是由电子工业协会(EIA，Electronic Industries Association)制定的数据终端设备和数据电路端接设备连接的物理接口标准，属于国际标准化组织 ISO 的开放系统互连(OSI)模型中的最低层，即物理层的协议标准。它规定了接口的机械、电气和功能特性。

RS-232-C 规定的机械特性是 25 针的插头/座，减去一些未定义的针外，实际上只定义了 20 根针的功能。用来连接两个设备至少要连接 3 根钱线，即信号地、发送数据和接收数据线。在采用 RS-232-C 连接计算机与终端的场合就只使用这 3 根线。

RS-232-C 的设计数据速率是 2000bit/s，连接设备间的距离也有规定。为了实现更高的数据速率和更远的距离连接，EIA 又制定了另一个 RS-422 标准。该标准的电气特性与 RS-232-C 不同，不用公共地，采用双线平衡传输的方式，在同样数据速率条件下，可达到较远的传输距离。在数据速率是 2000bit/s 条件下，连接设备间的距离可扩展到原有 RS-232-C 标准的约 80 倍。

【答案】A：②： B：① C：② D：④ E：⑦

第五章 专业英语试题精解

根据考试大纲的要求，高级程序员应具有大学毕业程度的英语词汇量，能正确阅读和理解计算机领域的英文文献，相当于大学四级英语水平。

解答专业英语试题的关键在于熟悉相关的计算机专业英语。试题以考查词汇为主，兼考语法。复习时应多读一些计算机方面的英语时文，积累一些计算机专业词汇。解题时，一般先考虑语义，后考虑语法。

1990～2000 年英语试题涉及的主题分布情况见表 5-1。总的来说，大多涉及流行的软件技术，时代感很强。

表5-1 历年高级程序员级专业英语试题统计

试题	所涉及的主题
1990年试题12、13	程序设计语言（programming languages）、集成软件（integrated software）
1991年试题12、13	二进制数（binary numerals）、计算机文化（computer literacy）、
1992年试题12、13	RISC与CISC、AI（人工职能）
1993年试题12、13	移动式计算机（Mobile computers）、C语言编码风格
1994年试题11、12	面向对象的数据库管理系统（Object-oriented DBMSs）、Windows NT
1995年试题11、12	微内核技术（microkernel technology）、窗口程序开发（Windows development）
1996年试题12、13	局域网技术（LAN technology）、Java语言
1997年试题12、13	关系数据库模型（relational database model）、多媒体（multimedia）
1998年试题12、13	拼写检查（spell checker）、局域网（LAN）
1999年试题14、15	信息产业（information sector）、网络管理工具（network management tools）
2000年试题14、15	VOIP、防火墙（firewall）

试题1（2000年试题14）

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

Network managers have long { A } practical voice-over-IP(VOIP) solutions. VOIP{ B } ease network management and decreases costs by converging at a company's telephony and data infrastructures into one network. And a VOIP solution implemented at a company's headquarters with far-reaching branch offices can { C }tremendous amounts of { D } in long distance phone bill, provided that solution delivers POTS-like voice { E } over the Internet.

供选择的答案

A：①await ②awaited ③awaiting ④awaits

B：①promise ②promised ③promises ④promising

C：①get ②put ③save ④waste

D：①cash ②money ③space ④time

E：①frequency ②length ③quality ④quantity

【解析】

题中A处考查语法,这里是现在完成时,动词形式是过去分词,应选择"awaited"。

B处也是个语法问题，这里是一般现在时，主语是单数形式，因而谓语动词的形式也是单数，应选择"promises"。

C处从语义上看是"节省"的意思，只有"save"一次符合要求。

D 处指费用、金钱，只有"money"一词适合。

E 处指语音质量，应选择"quality"。

参考译文：网络管理员一直期待着实用的 VOIP(IP 话音业务)解决方案。通过将公司的电话和数据基础设施集成于同一网络，VOIP 简化网络管理，降低成本。如果 VOIP 解决方案通过因特网提供类似 POTS 的声音质量，将其用于拥有无程分支机构的公司总部，可节省大量的长途话费。

【答案】A：② B：③ C：③ D：② E：③

试题 2 (2000 年试题 15)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

Basically, a firewall is a { A } process or a set of integrated processes that runs on a router or { B } to control the flow of networked application traffic { C } through it. Typically, firewalls are placed on the entry point to a { D }network such as the Internet. They could be considered traffic cops. The firewall's { E }is to ensure to ensure that all communication between an organization's network and the Internet conform to the organization's security policies.

供选择的答案

A：①dependent ②isolated ③related ④standalone

B：①browser ②client ③repeater ④server

C：①pass ②passed ③passing ④passes

D：①local ②national ③private ④public

E：①mole ②pole ③role ④rule

【解析】

题中 A 处显然是"独立的、单独的"的意思，应选择"standalone"。

从上下文看，B 处是指服务器，应选择"server"。

C 处则是个语法问题，现在分词作定语，应选择③。

从后面的"Internet"判断，D 处意义应当是"公共的"，只有"public"一词符合。

E 处意是"角色"的意思，选择"role"。

参考译文：基本上，防火墙是在路由器或服务器上运行的一个独立的或一组集成的处理器，用来控制通过网络应用程序流经它的信息流。典型的防火墙位于进入像因特网这样的公共网络的入口处。可以将它们看作是交通警察。防火墙的角色就是确保组织的网络与因特网之间的所有通信符合组织的安全政策。

【答案】A：④ B：④ C：③ D：④ E：③

试题 3 (1999 年试题 14)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

All of these applications will enhance the { A } of life and spur economic growth. Over half of the U.S. work force is now in jobs that are information { B } . The telecommunication and information sector of the U.S. economy now { C }for 12 percent of the Gross Domestic Product, groing much faster than any other sector of the economy. Last year the { D }in this sector exceeded 700 billion dollars. The U.S. exported over 48 billion doooars of telecommunication equipment { E }.

A：①quantity ②quality ③mass ④amount

B：①based ②bassed ③bases ④base

C：①look ②looks ③account ④accounts

D：①revenues ②expenses ③outputs ④loss

E：①among ②alone ③simple ④single

【解析】

题中 A 处显然指生活质量，应选择②。

B 处则是个语法问题，是被动语态。

C 处从语义上看是"占多少比例"，"account for"是习惯用法，注意一下语法，谓语动词是单数。

D 处指财政收入，只有"revenues"一词符合。

E 处是"单独"、"独自"的意思，语法上是副词作状态，选"alone"。

参考译文：所有这些应用软件会提高人们的生活质量，刺激经济增长。现在美国半数以上的劳动力所从事的工作依赖于信息。在美国经济中，电信和信息门类占国内生产总值的 12%，而且增长速度也比其他经济门类都快。去年该行业收入超过 7000 亿美元。美国电信设备的出口额超过 480 亿元。

【答案】A：② B：① C：④ D：① E：②

试题 4 (1999 年试题 15)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

The growth of switching has { A } a new generation of network management tools that help { B }cope with the challenges, { C }them to correlate device alarm in order to { D }pinpoint root causes, or to monitor service levels without depending on IP subnetting schemes. These new tools have become essential to coping with the primary { E }effect of any treatment that significantly increases network flexibility: added complexity.

供选择的答案

A：①create ②creates ③created ④creating

B：①technicians ②workers ③salesmen ④professors

C：①enjoying ②enabling ③engaging ④enriching

D：①better ②well ③welly ④goods

E：①back ②side ③front ④lateral

【解析】

本题是关于网络管理技术的，难度大一些。

A 处主要考语法，完成时动词形式。

B 处考词义，只有"technicians"(技术员)最合适。

C 处从语义和语法上看应是"enabling"，意思是"使……能"。

D 处副词修饰动词不定式，而且隐含比较级，只有"better"合适。

E 处是"副作用"的意思，显然用"side"。side 作形容词，表示"旁边的、侧面的、副的、枝的"，side effect 则指副作用。

参考译文：交换技术的发展产生了新一代网络管理工具。这些工具有助于技术人员应付挑战，使它们能与设备报警相联以更好地查明根本原因，或者不依赖于IP 子网配置来监控服务级别。这些工具对解决因显著增加网络适应性(额外的复杂性)带来的主要副作用的问题是必需的。

【答案】A：③ B：① C：② D：① E：②

试题 5 (1998 年试题 12)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

Many word processing programs include spell checker. It checks the spelling of every word in a { A } by looking up each word in its dictionary. If the word does not appear in the dictionary, the user is { B }to a possible misspelling and possible corrections are often { C }. Spell checker does not recognize unusual people names or specialized terms, but it will often allow you to to create your own personal dictionary of specialized words you often use. Spell checker is a valuable aids to proofreading, but it can not catch the { D }of one correctly spelled word for another (such as form for from). Thus it does not { E }a document is free of spelling errors.

A;①document ②equipment ③program ④statement

B、C ①alerted ②alternate ③guessed ④guided ⑤suggested ⑥surprised

D; ①addition ②condition ③notation ④substitution

E; ①committee ②correct ③grarantee ④prove

【解析】

参考译文：许多字处理都包括了拼写检查的功能，拼写检查通过查自己的字典来检查一个文件中的每一个单词。如果字典里没有这个单词，用户就会被警告可能拼写错误，同时也经常提供一个纠正该错误的建议。拼写检查程序不能识别不常用的人名或专用术语，但是它允许你生成自己的个人字典，把自己常用的词语加进去。拼写检查程序对文字校对来说是一个有用的工具，但是它不能检查到一个单词被另一个拼写正确的单词所代替这样的错误，因此它不能保证一个文件中没有拼写错误。

【答案】A：① B：① C：⑤ D：④ E：③

试题 6 (1998 年试题 13)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

A local-area network(LAN) is a communications network that { A } a variety of devices and provides a { B }for information exchange among those devices. The scope of the LAN is small typically a single building or a cluster of builings. The LAN is usually owned by the same organlization that owns the { C } devices.The internal data rate of LAN is at least several Mbit/s. The basic of LAN communciation is broadcasting. At each station, there is a transmitter/receiver that commuinciates over a { D }shared by other stations. A transmission from any one station is { E }to an received by all other stations.

供选择的答案

A、B、D、E：①broadcast ②cable ③control ④interconnects
⑤internet ⑥meander ⑦means ⑧medium
⑨modem ⑩output 11. switch 12. relay

C：①adopted ②attached ③selected ④unified

【解析】

参考译文:局域网是将许多设备互连在一起并提供一种在这些设备间交换信息的方法的通信网。局域网的范围通常比较小，如一个建筑物或一群建筑物。局域网通常为拥有那些连接在网上的设备的同一组织所拥有。局域网的内部传输率至少为几兆字节每秒。局域网的基本通讯方式为广播方式。每个站都有一个转发/接收器，通过一个与其他站共享的媒介来通讯。从任何一个站发送的消息都被广播发送到所有的站，被所有的站接收。

【答案】A：④ B：⑦ C：② D：⑧ E：①

试题 7 (1997 年试题 11)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

The relational database model requires the data be { A } through programs that don't rely on the position of the data in the database. This in in direct { B } to the other database mode, where the program has to follow a series of pointers to the data it seeks.{ A } program { C } a relational database simply asks for the data it seeks; the DBMS performs the necessary searches and { D }the information. The { E }on how the search is done are specific to the DBMS and vary from product to product.

供选择的答案

A：①accessed ②moved ③read ④wrote

B：①conduct ②contract ③contrast ④construct

C：①consulting ②containing ③querying ④queuing

D：①erases ②provides ③proves ④values

E：①details ②documents ③tails ④tenants

【解析】

参考译文：在关系型数据库模型中，要求通过程序访问数据时，不必依赖于数据在数据库中的位置。这种方式与其他数据库模式中采用的方式的不同之处在于：其他数据库模式中，程序需要通过访问一系列指针才能查到所要访问的数据。在关系型数据库中，程序查询数据库的方式只是直接指向所要查找的数据，数据库管理系统会实施必要的查询工作，然后将查询结果提供给程序。具体如何进行查询，与数据库管理系统有关，并且不同的数据库产品之间也有所不同。

【答案】A：① B：③ C：③ D：② E：①

试题 8 (1997 年试题 12)

从供选择的答案中，选出应填入下面叙述中{ }内的最确切的解答，把相应编号写在答卷的对应栏内。

When most people refer to multimedia, they generally mean the combination of tow
or more continuous media, usually with some user { A } . In practice, the two media are normally audio and video, that is, { B }plus moving { C }.

It should be obvious by now that transmitting multimedia material in uncompressed form is completely out of { D }, the only hope is that massive compression is possible. Fortunately, a algorithms that make multimedia transmisson { E }.

供选择的答案

A～C：①display ②games ③help ④interaction
⑤pictures ⑥sound ⑦web

D、E：①impossible ②fearful ③feasible ④program
⑤questio ⑥thing

【答案】A：④ B：⑥ C：⑤ D：⑤ E：③

试题 9 (1996 年试题 13)

从供选择的答案中，选出应填入下面叙述｛｝的最确切的解答，把相应编号写在答卷的对应栏内。

｛ A ｝ whether to go with a shared or switched fast LAN technology once you start to maxout your shared Ethernet of token-ring LAN is tough call. Users and analysts involved in ｛ B ｝high-speed LANs have learned the benefits of each. However there are trade-offs to each, and ｛ C ｝ which fast LAN technology to choose is critical for ｛ D ｝network performance problems and ｛ E ｝costly purchasing mistakes.
供选择的答案

A～E：①avoiding ②clipping ③choosing ④deciding ⑤dissolving ⑥ensuring ⑦evolving ⑧implementing ⑨knowing ⑩solving

【解析】

参考译文：你一旦想扩大你自己的以太网或令牌环 LAN 的共享能力，就面临着一种选择：是用共享快速 LAN 技术还是交换快速 LAN 技术?那些已决定高速 LAN 的用户们和分析家们都知道，这两种技术各有其优点也都有其不足。明确选择哪一种技术是解决网络性能问题和避免购买浪费的关键。

【答案】A：④ B：⑧ C：⑨ D：⑩ E：①

试题 10 (1996 年试题 14)

从供选择的答案中，选出应填入下面叙述中｛ ｝内的最解切的解答，把相应编号写在答卷的对应栏内。

The most accurate and most boring way to ｛A｝ Java is that is a new computer programming language developed by Sun Microsystems that creates ｛ B ｝independent programs the that can be distributed and run remotely. To run Java programs, a computer must have a Java interpreter. Currently ｛ C ｝Java programs are small″applets″ that are ｛ C ｝as part of web pages.

When you use a Java ｛ D ｝browser to view a Web page that in cludes a Java applet, the broser loads the applet onto your computer through your modem or network. Then the Java interpreter runs the applet, which could include animation or sound, on your computer rather than transmitting the code bit by bit over the Internet. A few thousand bytes of Java code can turn into a powerful program on your computer.
So an applet could in clude ｛ E ｝with Java interpreter.

供选择的答案

A：①command ②comment ③describe ④discover

B：①condition ②platform ③programmer ④workstation

C、D：①available ②capable ③possible ④probable
⑤stable ⑥valuable

E：①animation or sound ②animation and sound
③animation and be run ④animation and be loaded

【解析】

参考译文：关于 Java 一词最准确且最乏味的描述是，它是一种新的计算机程序设计语言，是由 Sun Microsystems 公司开发的，它用来生成与平台无关的可以分布在远程机器上运行的程序。要想运行 Java 程序，计算机上必须装有 Java 解释器。当前可用的 Java 程序是一些小(应用程序)片断，被用作 Web 页的一部分。

当你用一个支持 Java 的浏览器来浏览一个包含 Java 小应用程序的 Web 页时，浏览器将这个小应用程序通过调制器或网络传到你的计算机上，然后 Java 解释器运行这个小应用程序，其中可能镶嵌动画和声音，这些操作都是在你的计算机上完成的，而不是通过 Internet 一位一位地传输代码过来。几千字节的 Java 代码可能在你的计算机上变成一个强有力的程序。
所以一个 Java 小应用程序可以包含动画，需要 Java 解释器来执行它。

【答案】A：③ B：② C：① D：② E：③

试题 11 (1995 年试题 11)

从供选择的答案中，选出应填入下面英文语句叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

For users, microkernel technology promise { A }, compact, and sophisticated operating systems that are typically { B }across a range of hardware platforms. These operating systems will be customizable to let users run multiple operatin system and application "per-Sonalities" on top of a single microkernel {C}.Microkernel-based systems can provide this flexbility because the core operating
system functionc are { D } from the large number of utilities, features and extension that are layered on top of them. As a result, updation and maintaining operating system is easier, since developers don't have to modify the { E } every time they must add a new feature.

供选择的答案：

A：①slow ②fast ③quickly ④speed

B：①moveable ②made ③portable ④use

C：①bases ②foundations ③systems ④foundation

D：①separate ②connected ③compared ④selected

E：①core ②center ③heat ④utility

【解析】

参考译文： 对用户来说，微内核技术有希望实现能在许多硬件平台上移植的快速、紧凑和先进的操作系统。这些操作系统可以定制化，以使用户可在单个微内核基础上运行多个操作系统和应用程序。

基于微内核的系统之所以能够提供此灵活性是因为核心的操作系统功能与大量用程序、特征和扩展功能是分离的。它们分层加于核心功能上，其结果是使更新和维护操作系统更加容易了。因为开发人员不必在每次加入一个新特色时都修改核心。

【答案】A：② B：③ C：④ D：① E：①

试题 12 (1995 年试题 12)

从供选择的答案中，选出应填入下面英语文句叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

Application development increasingly means Windows development, and the popularity of visual development tools has { A } in tandem with Windows itself. These tools create beautiful windowing { B } , and their fast development cycles and easy learning curves make them a good { C } for many types of PC development projects. Today's developers are leveraging these tools and the abundance of cheap, powerful PCs to shift the balance of power to the desking.
As the world moves inexorably toward Windows and other { D }user interfaces, developers can choose from an abundance of { E } oriented tools. Popular examples in clude Microsoft Corp's Visual Basic, Powersoft Corp's PowerBuilder, Gupta Technology Corp's SQL Windows, and so on.

供选择的答案：

A：①rise ②risen ③rised ④rising

B：①interfaces ②pictures ③graphs ④books

C：①choose ②selections ③choice ④select

D：①graphics ②graph ③graphitic ④graphical

E：①visually ②lively ③quickly ④specially

【解析】

参考译文：应用程序的开发越来越多地意味着 Windows 的开发。直观化的开发工具的流行紧随着 Windows 本身的普及而发展。这些工具能生成漂亮的窗口式接口，其快速开发周期和易于学习的过程使它们成为多种 PC 开发项目的良好选择。今天的开发人员正利用这些工具和大量廉价而功能强有和 PC 机把力量的平衡移到台式机上。

随着世界坚定地朝着 Windows 和其他图形用户接口转移，开发人员可以在大量的面向直观化的工具中进行选择。流行的例子包括 Microsoft 公司的 Visual Basic、Powersoft 公司的 Power Builder, Gupta 技术公司的 SQL Windows 等。

【答案】A：② B：① C：③ D：④ E：①

试题 13 (1994 年试题 11)

从供选择的答案中，选出应填入下面英语文句叙述中{ }内的正确答案，把编号写在答案的对应栏内。
Object-oriented DBMSs integrated a variety of { A } data types-such as business procedures, graphics, pictures, vioce and annotated text.
Object orientation also makes a { B }to application development efficiency.
It makes the data, functions, attributes, and relationships an integral part of the { C } .
In this way, objects can be reused an replicated.
Some leading RDBMS vendors support the concept ofintegrating object management capabilities with their current line of relational products.
That capability enable users to { D }the development cycle, since integrity logic and business rules no longer need to be programmed { E } each application.

供选择的答案

A～E：①tool ②in ③idea ④contribution
⑤joke ⑥short ⑦object ⑧theoretical
⑨extent ⑩shorten 11.real-world 12.into

【解析】

参考译文:面向对象的数据库管理系统将各种现实世界的数据类型--商业事务过程、图形、图片、语音和注释文本等加以集成。

面向对象对应用开发效率也有贡献。它使数据、函数、属性和关系成为对象的完整的部分，这样，对象就可以被重新使用和复制。

某些领先的关系数据库管理系统的销售商支持将对象管理能力与他们目前经营的关系产品系列相结合的概念。这种能力可以使用户缩短开发周期，因为完整的逻辑和事务规则不再需要为每个应用程序编程。

【答案】A：11 B：④ C：⑦ D：⑩ E：12

试题 14 (1994 年 12)

从供选择的答案中，选出应填入下面英语文句叙述中{ }内的正确答案，把编号写在答卷的对应栏内。

In { A }software, one question of near-universal interest is how rapidly Windows NT, which began shipping last summer, will be { B } in the marketplace and for what uses. For the most part, observers are { C } about the operating system's long term prospects, but expect it to remain on { D } rather than desktops during 1994, saying that it won't become a mainstream product until { E } PCs typically have 16 to 24MB of RAM, which is unlikely to happen next year.

供选择的答案

【答案】A～E：①applications ②accepted ③clients ④enthusiastic ⑤Servers ⑥desktop ⑦systems ⑧replaced

【解析】

参考译文：在系统软件中，一个普遍关心的问题是去年夏天开始出售的 Windows NT 将会如何快速地被市场所接受以及起什么作用。大部分观察家们对该操作系统的长远前景表示出热情，但预料在 1994 年间它仍将用在服务器上而非台式机上。他们说，在典型的台式 PC 机具有 16MB 至 24MB 内存之前，它不会成为主流产品，而这件事明年是不大可能发生的。

【答案】A：⑦ B：② C：④ D：⑤ E：⑥

试题 15 (1993 年试题 12)

从供选择的答案中，选出应填入下面英语文句中{ }内的正确答案，把编号写在答卷的对应栏内。

Mobile computers- which { A } laptops, notebooks, subnotebooks and handhelds - { B } so ubiquitous in such a short time, no sunrise to hear who say:"It will define the leading edge { C } the next five years or so." The most remarkable { D }mobile computers is the amount of data storage and memory packed { E }their tiny boxes. These devices not only handle windows easily but also run storage-hungry programs.

供选择的答案

A、B：①included ②include ③have included ④have become
⑤had become
C～E：①from ②for ③about ④at
⑤on ⑥into

【解析】

参考译文：移动式计算机包括膝上型、笔记本型、小笔记本型和掌上型，在如此短的时间内变得如此之普遍，不必吃惊它将在下一个 5 年或更长时期名列前茅。移动式计算机最显著的特点是装入其小盒子的数据存储的量。这些设备不仅能容易地处理窗口作业而且能运行大存储量的程序。

【答案】A：② B：④ C：② D：③ E：⑥

试题 16 (1993 年试题 13)

从供选择的答案中，选出应填入下面英语语文句中{ }内的正确答案，把编号写在答卷的对应栏内。

One of the guidelines in writing the C code for a software tool is as follows: write code that is as clear and as simple as { A }. The C language can be difficult to read if you combine all { B }features in a single statement. Break complicated { C }into several easy to understand statements for the { D }of readability. This style helps to make your programs more { E } and error-free.

供选择的答案

A～E：①readable ②reusable ③Possible ④semantic
⑤syntactic ⑥constructions ⑦safe ⑧impossible
⑨sake ⑩structure

【解析】

参考译文：为软件工具写 C 程序源码的一个规则如下：写的程序尽可能清晰、简单。如果你把所有句法特征组合在单个语句中，C 语言将会很难读。为了提高可

读性应打破复杂结构，将其分为几个易于理解的语句。这种风格有助于使你的程序更加可读，而且不出错误。

【答案】A：③ B：⑤ C：⑥ D：⑨ E：①

试题 17 (1992 年试题 12)

从供选择的答案中，选出应填入下列英语文句中{ }内的正确答案，把编号写在答卷的对应栏内。

In recent years, one of the more poplar topics panel discussions at computer conferences and trade { A } has been the"RISC versus CISC"debate.

RISC processors feature a small number of instructions that each executes in { B } machine cycle. CISC processors use complex instructions that can take several cycles to execute.
The RISC Versus CISC debate won't be decided by panel discussion 1 it will be won in the marketplace. And the deciding factor may have little to do with { C } of instructions and registers) and more to do with parallelism.

Since their conception, RISC processors have been evolving toward microparallelism, in corporating parallel-processing features { D } the processor. RISC processors feature pipelining, whereby many instructions can be decoded while one instruction executes.
RISC processor, however, are moving to toward pipelines for each unit of the processor.
CISC processors also employ pipelining. They have many interger instructions that execute in one cycle, but the varying execution times of CISC instructions { E } the effectiveness of parallelis.

供选择的答案

A～E：①union ②two ③numbers ④between ⑤limit
⑥contents ⑦shows ⑧one ⑨within ⑩enhance

【解析】

参考译文：近年来，在计算机学术会议和贸易展览会上最流行的会场讨论课题之一是"RISC 对 CISC"的争论。

RISC 处理器以少量指令为特征，且每条指令在一个机器周期中执行。CISC 处理器使用复杂的指令，它们可能花费若干周期来执行。

RISC 对 CISC 的争议将不由会场讨论来作结论，它将在市场中定胜负。并且，决定的因素也许很少取决于指令和寄存器的数目，而更多取决于并行性。

自从 RISC 概念提出后，RISC 处理器正向着微并行性发展，它在处理器内部结合了并行处理的特性。RISC 处理器以流水线为特点，在流水线上当一条指令执行时，许多指令能被解码。况且，RISC 处理器正向着每个处理器单元多流水线发展。

CISC 处理器也使用流水线。它们有许多在一个周期中执行的整数指令，但是 CISC 指令变化的执行时间限制了并行性的效率。

【答案】A：⑦ B：⑧ C：③ D：⑨ E：⑤

试题 18 (1992 年试题 13)

从供选择的答案，选出应填入下列英语文句中{ }内的正确答案，把编号写在答卷的对应栏内。
While most recemt attemtion in the AI field has been focused on expert system software, AI { A } has also seen dramatic advances. Activity, In the past years was characterized by new low-cost, powerful LISP machines, the introduction of AI workstations, LISP compilers becoming available for all major professional and engineering workstations, and the personal computer emerging as a { B }tool for expert system development. The next few years will see this technology evolves further.
Because the{ C } of an AI computer represents a sizeable investment, companies should carefully { D } all options that are available as well as have a good idea of what the next generation of systems will offer in order to { E }the optimum system. This publication provides the Information necessary to gain this understanding.

供选择的答案

A～E：①choice ②read ③importan ④software
⑤hardware ⑥significant ⑦emergence ⑧survey
⑨purchase ⑩select

【解析】

参考译文：当 AI 领域最近的注意力集中于专家系统软件时，AI 硬件也显示出巨大的进展。过去一些年的活动有下述特征：新的低价格和强功能的 LISP 机、引进 AI 工作站、LISP 编译器可用于所有主要的专业或工程工作站以及个人计算机作为专家系统开发的值得注意的工具出现。最近几年将会看到这个技术的进一步演变。

因为购买一个 AI 计算机意味着一笔相当在的投资，公司为了选择一个最佳系统应仔细地考查所有可用的选择，并对下一代系统将提供什么有一个好的想法。本文对了解这些东西提供了必要的信息。

【答案】A：⑤ B：⑥ C：⑨ D：⑧ E：⑩

试题 19 (1991 年试题 12)

从供选择的答案中选出应填入下列英语文句中{ }内的正确答案，把编号写在答卷的对应栏内。
It is traditional when dealing with languages of all sorts to try to separate co
nerns with{ A } , the subject of syntax, from concerns with B , the field of semantics, Consider the simple "language" of binary numerals. Some examples of binary mumerals are

```
    0      1
  101   0101
11001 101111
```

A communication in this language evidently consists of a finite sequence of characters '0', '1' This is just syntax however, and says C about what such a communication is intended to mean.
Mumbers are "D" mathematical concepts, whereas the digit strings that strings that appear on paper are numerals, that is to say, E representations or descriptions of numbers.

供选择的答案

A～C：①symbol ②form ③meaning ④context
⑤nothing ⑥concept
D、E：①abstract ②concrete ③simple ④ordinary
⑤symbolic ⑥logic

【解析】
参考译文：处理各种语言时，传统的做法试图将有关形式的语法问题和有关含义的语义方面区分开来。考虑简单的二进制数的"语言"。下面是二进制数的例子

【答案】 A：② B：③ C：⑤ D：① E：⑤

试题 20 (1991 年试题 13)

从供选择的答案中选出应填入下列英语文句中{ }内的正确答案，把编号写在答卷的对应栏内。
With the widespread use of the personal computer, many authorities in the field of { A } have pointed out the need for computer literacy.
Unfortunately, there is no { B } agreement as to what the term "computer literacy"
means. Some feel that computer literacy means knowing how to make the

computer "computer literacy" means. Some feel that computer literacy means knowing how to make the computer "compute"; that is, knowing how to program computers in one or more programming languages.
Others feel that knowing how to program is merely a small segment of computer literacy. These people { C } the major emphasis in schools should be on teaching how to effectively use the many software packages that are available.
Still others suggest that computer literacy education is not required. They suggest tha computers are being so rapidly integrated into our society that using a computer will be as { D }as using a telephone or a video tape recorder, and that special education will not be necessary.

{ E } of one's definition of computer literacy, it is recognized by most that learning to use a computer is indeed an important skill in modern society.

供选择的答案

A: ①culture ②science ③education ④industry

B、D: ①equal ②universal ③different ④difficult
⑤common ⑥big

C: ①claim ②deny ③define ④call

D: ①Importance ②Instead ③Because ④Regardless

【解析】

参考译文：随着个人计算机的广泛使用，许多教育领域的权威人士已经指出了对计算机文化的需要。

不幸的是对于"计算机文化"术语的含义尚无一致的意见。某些人觉得计算机文化意味着知道如何用计算机来"计算"；也就是说，知道如何以一种或多种编程语言为计算机编程。

另一些人认为如何编程只是计算机文化的一小部分。这些人主张学校中的教学重点应是教会学生如何有效地使用大量已有的软件包。

另外一些人建议计算机文化教育是不需要的。他们建议计算机正如此快速地进入我们社会中，以致于使用计算机将像使用电话或录像机一样普通，特设的教育是不必要的。

抛开人们关于计算机文化的定义，大多数人承认学会使用计算机的确是现代社会中的一项重要技能。

试题 21 (1990 年试题 11)

将下列英语译成中文(只可使用词典)：
Programming languages
Ten years ago the proliferation of programming languages caused many people to foresee the development of a computer-age babel where, in total ignorance of every other language, each programmer would learn only his own chosen language. That unhappy situaton has not occurred for several reasons. First, effective efforts have been made to standardize particular languages such as Fortran and Cobol. It should be pointed out that pragmatic rather than scientific considerations motivated this standardization movement. However, the second reason that Babel has been averted is that computer scientists have begun to apply the scientific method to organize the classification, comparison, and appreciation of various programming languages.
Due to the efforts of McCarthy(1962), Landin(1964), Strachey(1996), Wegner(1968), and others who provided insight into operational models of computation. We can now evaluate programming languages in terms of an unifying view of computation structures. Semantics and the expressive power resulting from modularity can now be studied in terms of the data structures and the accessing paths to them established during the execution of the control statements of the language.

【答案】

参考译文：程序设计语言

十年前程序设计语言的激增使得许多预言在 Babel 时代，计算机空想通天塔的发展情形(Babel 是基督教《圣经》中的城市名，诺亚的后代拟在此建通天塔，上帝怒其狂妄，使建塔人突操不同语言，塔因此终未建成。这里，用来比喻使用多种程序设计语言的空想)，在那里每个程序员完全可以忽略其他任何的语言，仅学习他自己选择的语言。由于种种原因，这种不幸的状况未发生。首先，有效的努力对特定的语言如 Fortran 和 Cobol 等的标准化起了作用。应指出，实用的而不是科学的考虑推动了这个标准化的进程。然而，Babel 空想得以避免的第二个原因是计算机科学家已开始应用科学的方法来组织各种程序设计语言的分类、比较和鉴别。

通过 McCarthy(1962)、Landin(1964)、Strachey(1996)、Wegner(1968)和其他对计算的操作模型深入考查的人们的努力，现在我们正以统一的计算结构观点来评价程序设计语言，可以通过数据结构以及在执行语言语句时建立的访问路径，来研究语言和模块化导出的表达能力。

试题 22 (1990 年试题 12)

将下列英语译成中文(只可使用词典)：

Integrated Software
Convenience and saved time, work, and effort are the promises of integrated software.
The antithesis of stand-alone packages, integrated software delivers a collection of applications based upon a common user interface and sharable data.
In its most common form, the integraged product includes a word processor, a spreadsheet, and some form of database. Many packages add telecommunications, presentaion graphics, and outline modules.
Comprehensive products throw in desktop accessories such as calculators, calendars, DOS shells, and other utilities.
Even when stand-alone products are from the same vendor, it can be frustrating trying to move information between applications or simply trying to remember which key to press to call up the menu. That is why integrated packages appeal to many users, particularly novices. Using an integrated product saves you the headache of trying to move data in a Brand X word processor to a Brand Y spreadsheet. And because the integrated package is a single product from a single vendor, training, support, and upgrades also are made simpler.

【答案】
参考译文：集成软件

方便、省时、省工县有效是集成软件的应具备的性质。

同独立软件包相比，集成软件提供一组基于公共用户界面和共享数据的应用程序。

在其大多数常用的形式中，集成的产品包括字处理、电子报表和某种形式的数据库。许多软件包加进了远程通信、表示图形以及轮廓模块。复杂的产品增添了如计算机器、日历等办公辅助软件、DOS 外壳及其他实用程序。

即使独立的产品出自同样的供应商，试图在应用程序间传递信息或者只是简单地记住按哪个键来调出菜单都可能行不通。这就是集成软件包为什么对许多用户特别是新用户有吸引力的原因。使用集成的产品能省去试图将牌号 X 字处理程序中的数据传到牌号 Y 的报表程序这类令人头痛的问题。同时，因为集成软件包是来自单一供应商的单一产品，培训、支持和升级也变得更简单。

5. 软件设计试题精解

第六章 软件设计试题精解

根据考试大纲的要求，高级程序员不仅要具备高水平的程序编制能力，而且要熟练掌握软件设计的方法和技术，具备一定的软件设计能力。软件设计能力由下午试题考查，目前是 3 道题中选答 2 道。试题的形式一般是根据有关的软件分析设计图(常见的有数据流图、程序流程图、系统流程图和 ER 图)和其他说明资料，考生按要求回答问题或填空。试题内容涉及到流程图设计、1 软件界面设计、数据库设计和软件测试等多个方面，要求考生熟练地掌握软件分析和软件设计的常用方法和技术。

6.1 2000 年度软件设计试题解析

2000 年度下午试卷共有 3 道软件设计试题，都涉及到流程图，有两个突出的特点：

第一，试题 1 的问题 2 要求写出操作的内容，试题 2 的问题 2 和问题 3 要求写出子程序的功能和操作内容。这两个问题的形式在历年试题中比较少见，但这也正反映了对高级程序员软件分析能力的要求是一种科学的考查方式。

第二，在试题 3 的问题 1 中，要求找出哪张图中的哪些文件不必画出。这种形式的问题在历年试题中也比较少见，而且对考生的要求也比较高，要求考生仔细研究每一张图。在这种问题中，解题的指导原则与解答其他数据流图试题相同。只要把握好指导原则，问题的解答还是有规律可循的。

除了上述两个方面，其他关于数据项的定义、判断条件的填写以及流程图处理具体个例的考
查方式在历年试题中比较多见，就不再分析了。

试题 1 (2000 年试题 1)

阅读以下说明和流程图，如图 6-1 所示，回答问题 1 和问题 2，将解答写在答卷的对应栏内。

[说明]

本流程图实现由成绩文件生成学生成绩一览表。

某中学某年级的学生成绩数据(分数)登录在成绩文件 F0 中，其记录格式如下：

学号	姓名	课题 1 成绩	课程 2 成绩	……	课程 6 成绩

由该成绩文件生成如下表所示的学生成绩一览表，并按学号升序排列。表中的名次是指该生
相应课程在年级中的名次。

学号	姓名	课程 1		课程 2		……	……	课程 6	
		成绩	名次	成绩	名次	……	……	成绩	名次

在如图 6-1 所示的流程图中，顺序文件 F0 是学生成绩文件。F0 文件经处理 1 处理后产生顺序文件 F，然后经过处理 2 至处理 4 对文件 F 进行处理和更新。在处理 5 中，仅对文件 F 的记录进行编排，输出学生成绩一览表，但不进行排序和增加名次等处理。

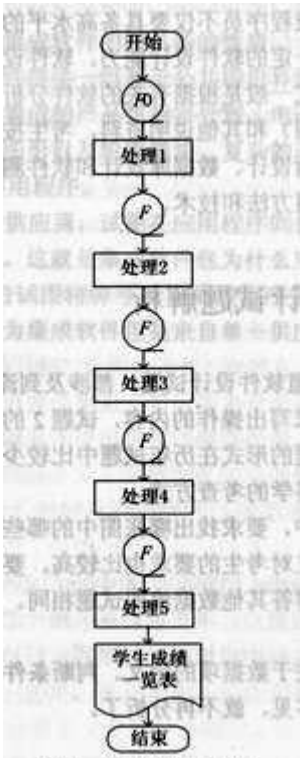


图 6-1 2000 年试题 1 流程图

[问题 1]

流程图中文件 F 的记录格式设定为如下形式：

学号	姓名	课程代号	①	②

其中的①和②应定义的为何种数据项?

［问题 2］

简述处理 2、处理 3 和处理 4 作何种处理，若有排序处理则需指明排序的键和序(升序或降序)。

【解析】

本题有一定的难度。在给出了文件 F0 和最后输出的学生成绩一览表后，整个流程图的主体部分(处理 2、处理 3 和处理 4)的内容都需要根据试题说明和流程图的结构进行安排。对于此类试题，需要比较系统的输入数据和输出数据，以确定从输入数据转换成符合输出要求的数据需要经过何种处理，然后再合理安排这些处理的顺序。

系统输的成绩文件 F0 是一个顺序文件，在学号、姓名之后顺序存放了 6 门课程的成绩。文件 F0 经过处理 1 处理后，产生顺序文件 F，然后经过处理 2、处理 3 和处理 4 的处理和更新。在处理 5 中，对文件 F 的记录进行学生成绩一览表的编排输出，不进行排序和增加名次的处理。关于文件 F，试题说明中给出了部发记录格式，除学号、姓名和课程代码外，还有两个未定义的数据项，需要根据实际情况给出。

研究学生成绩一览表的格式，我们可以看到，学号按升序排列，课程也有一个排列次序，而且增加了某学生某门课题成绩的名次。这就要求对每门课程的成绩进行排序，然后按照排序的结果给出名次。这样，在处理 4 处理文件 F 的过程中，需要按学号排序，同时还要按课程代码排序。即使这不是处理 4 的全部内容，也是处理 4 的一部分内容。

如果需要从文件 F 编排输出学生成绩一览表，会发现在文件 F 的记录格式中还有两个未定义的数据项，比较文件 F 的记录格式和学生成绩一览表，发现学生成线一览表中的成绩和名次两个数据项在文件 F 的记录格式和学生成绩一览表，发现学生成绩一览表中的成绩和名次两个数据项在文件 F 的记录格式中没有反映，先假定这里的两个未定义数据项就是成绩和名次。

或许有的考生要问，如果在处理过程中不产生名次，在处理 5 中只是按排序的顺序来输出是否可以呢?也就是说，在处理 5 中，文件 F 是按课程排序的，同时每门课程又按成绩排序。这是不可能的，由于每个学生各门课程成绩的名次各不相同，如果没有名次这个数据项，是无 法同时保证学号和成绩的有序性的。如果不能够同时保证学号和成绩的有序性，处理 5 编排输出效率将会大大降低。

基于以上的分析，必须明确名次这个数据项应该怎样加入。如果要产生名次，需要对成绩进行排序，这是常识。由于文件 F 的记录格式中提供了数据项名次，可以设想，在每门课程按成绩排序后，再按课程代码和成绩向文件 F 中写入名次是非常容易的。至此，从处理 1 后的文件 F 到满足上述需求的数据已经不需要额外的操作了。

结合以上的分析，试着在头脑中模拟一遍程序的运行过程，以确认上述分析的正确性。下面的工作是将以上的分析用准确的语言描述出来，并进行合理的划分，把它们安排到处理 2、处理 3 和处理 4 中去。

这里按照流程图的顺序来分析，而不是像上面那样逆流程图结构进行分析。在处理 1 后，需要对文件 F 进行排序，包括按课程代码排序(升序或降序排列都可以)和按成绩(升序或降序都可以)排序。排序后，需要在文件 F 中按照排序结果写入名次，然后，将文件 F 再按学号(必须是升序)和课程代码(升序和降序都可以)排序。

将以上所描述的 3 个部分内容分配到处理 2 至处理 4 中，再通读流程图，以确认在该题的解答过程中没有遗漏重要的信息内容。

【答案】
[问题 1] ①成绩 ②名次
或①名次 ②成绩

[问题 2]

处理 2：以课程代码(升序)和成绩(降序或升序)为键对文件 F 排序。

处理 3：对每个课程代码，决定学生名次，并写入文件 F 的相应字段。

处理 4：以学号(升序)和课程代码(升序)为键对文件 F 排序。

注：在处理 2 和处理 4 中，课程代码的排序方式也可以同时为降序。

试题 2 (2000 年试题 2)

阅读以下说明和流程图，回答问题 1 至问题 4，将解答写在答卷的对应栏内。

【说明】

本流程力量(如图 6-2 所示)是将中缀表示的算术表达式转换成后缀表示，如中缀表达式

(A-(B*C+D)*E)/(F+G)

的后缀表示为：

ABC*D+E*-FG+/。

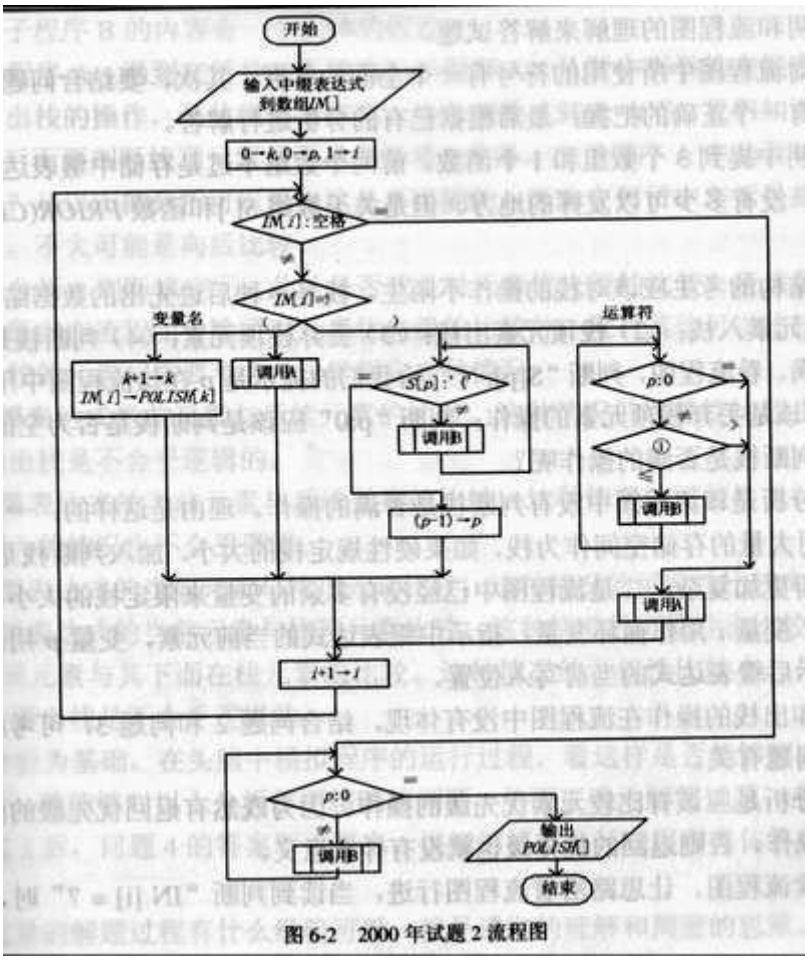


图 6-2 2000 年试题 2 流程图

为了方便，假定变量名为单个英文字母，运算符只有+、-、*和/(均为双目运算符，左结合)，并假定所提供的算术表达式非空且语法是正确的。另外，中缀表示形式中无空格符，但整个算术表达式以空格符结束。流程图中使用的符号的意义如下：

数组 IN [] 存储中缀表达式；

数组 POLISH [] 存储其后缀表示；

数组 S [] 是一个后进先出栈；

函数 PRIOR(CHAR)返回符号 CHAR 的优先级，各符号的优先级如下表所示。

CHAR	PRIOR(CHAR)
*/	4
+-	3
(	2
)	1

[问题 1]

填充流程图中①的判断条件。

［问题 2］

写出子程序 A 的功能，并顺序写出实现该功能的操作。

［问题 3］

写出子程序 B 的功能，并顺序写出实现该功能的操作。

［问题 4］

中缀表达式

(A+B-C*D)*(E-F)/G)经该流程图处理后的输出是什么?

【解析】
本题是关于中缀表达式转换为后缀表达式的试题。如果考生的知识面比较宽，应该了解该种问题是有常用算法的。本题的流程图就是常用算法的一种描述，如果掌握了该算法，解答将是非常容易的。下面先用自然语言叙述该算法。

转换时，从中缀表达式的左端开始，向右逐个扫描处理。表达式包含 4 类元素：(1)变量名；(2)左括号'('；(3)，右括号')'；(4)运算符号。处理时需要用到后进先出的数据结构--栈。具体处理规则如下：

(1)如果当前元素为变量名，则直接将该元素写入后缀表达式。

(2)如果当前元素是左括号，则将该元素入栈。

(3)如果当前元素是右括号，则栈顶元素顺序出栈，写入后缀表达式，直到栈顶元素为左括号，就将该栈顶元素丢弃。

(4)如果当前元素是运算符号，则分两种情况：(a)当栈不空时，比较当前元素与栈顶元素的优先级(优先级从高到低分别为：*/、+-、(、))，当栈顶元素优先级当前元素的优先级时，当前元素入栈，否则栈顶元素顺序出栈，写出后缀表达式，直到栈空；(b)当栈空时，当前元素入栈。

(5)在处理到中缀表达式结尾时，将栈内元素顺序出栈，写入后缀表达式。

但是，如果考生对上述算法不甚了解甚至一无所知呢?下面假定考生不知道上述算法，仅凭对试题说明和流程图的理解来解答试题。

首先，要对流程图中所使用的符号有一个全面的掌握；其次，要结合问题，对流程图中各操作的含义有一个正确的把握；最后根据已有的分析进行解答。

在试题说明中提到 3 个数组和 1 个函数，前两个数组不过是存储中缀表达工和后缀表达式的数组而已，没有多少可以发挥的地方。但是关于数据组 S［］和函数 PRIOR(CHAR)，需要更为深入的理解。

熟悉数据结构的考生应该对栈的操作不陌生。栈是一种后进先出的数据结构，对栈共有 5 种操作：(1)元素入栈；(2)栈顶元素出栈；(3)丢弃栈顶元素；(4)判断栈是否为空；(5)判断栈是否已满。看流程图，判断"S［p］；'('"给我们的提示 p 在该流程图中用作栈顶指针；操作 p-1→p 应该是丢弃栈顶元素的操作；判断"P:0"应该是判断栈是否为空的操作。那么，进栈、出栈和判断栈是否满的操作呢?

第 1 步的分析是该流程图中没有判断栈是否满的操作。理由是这样的：一是问题规模较小，不需要占用大量的存储空间作为栈，如果硬性规定栈的大小，加入判断栈是否满的操作，会使流程图变得更加复杂；二是流程图中已经没有多余的变量来限定栈的大小，因为从流程图中可以看出，变量 i 用作循环环变量，指示中缀表达式的当前元素，变量 p 用作栈顶指针，变量 k 用作指示后缀表达式的当前写入位置。

但是进栈和出栈的操作在流程图中没有体现，结合问题 2 和问题 3，可考虑这两个操作是否与这两个问题有关。

第 2 步的分析是应该有比较元素优先级的操作。因为既然有返回优先级的函数，必定有比较优先级的操作，否则返回的优先级也就没有什么意义。

现在来阅读流程图。让思路沿着流程图行进，当读到判断"IN［i］=?"时，流程图分成了 4 个分支。

(1)当当前元素 IN［i］为变量名时，操作"IN［i］→POLISH［k］"显然是将变量名直接写入后缀表达式的操作。

(2)当当前元素 IN［i］为左括时，执行子程序 A。

(3)当当前素元 IN［i］为右括时，如果栈顶元素为括号，则丢弃栈顶元素；否则调用子程序 B，直到栈顶元素为左括号，将栈元素丢弃。

(4)当当前元素 IN［i］为运算符时，分为两种情况：(a)如果栈空，则元素入栈；(b)如果栈不空，则执行判断填空①，如果结果为大于号，则执行子程序 A，否则，执行子程序 B，直到栈空或判断填定①的结果为大于号时再执行子程序 A。

处理完一个元素后，循环变量 i 递加，继续处理下一个元素。处理完中缀表达式后，如果栈非空，就执行子程序 B，直到栈空，然后输出后缀表达式。

读完流程图后，就会注意到，子程序 B 总是与栈空联系在一起的。在栈的操作中，导致栈空的操作有两种，一是丢弃栈顶元素，二是栈顶元素出栈。子程序 B 中几

乎是不可能包含丢弃栈顶元素的操作的，比较中缀表达式和后缀表达式，可以看出在转换过程中消失的元素是左括号和右括号，而这部分操作在处理右括号时就做了(右括号不入栈，左括号被丢弃)。因此可以断定，在子程序 B 中包含的是栈顶元素出栈的操作。如果这里包含出栈的操作，那么出栈后的元素应该送到哪里去?显然，如果是运算符，应该写入 POLISH [] 中。在子程序 B 中会有对元素优先级的比较吗?结合以上的分析，从最后一次使用子程序 B 的情况看，不会有比较元素优先级的操作，因为中缀表达式已经处理完毕，现在需要的是将栈内元素全部出栈。这样对子程序 B 的内容有一个大体的假设。

现在看子程序 A，遇到左括号时需要执行子程序 A。从题中列举的中缀式表达式看，该子程序不能包含出栈的操作，具体理由不解释。结合判断"S [p]：'('"，我们知道这里包含入栈的操作，因为后面要判断栈顶元素是不是左括号。此外，在子程序 A 中是否可以包含对元素优先级的判断?从以上的分析可以看出这是不可能的，因为在例子中左括号是第一个元素，没有比较对象，不大可能是向后比较。

基于以上分析，判断填空①，此处是否就是对元素优先级的比较?应该是的，因为结合试题说明可以断定在流程图中除了对元素优先级的比较外，没有其他可以比较大小的对象。但具体参加与比较的元素却需要考虑，这里假定 5 种情况：

(1)中缀表达式当前元素与后继元素的比较。这种情况的可能性比较小，在流程图中后续元素影响出栈是不合乎逻辑的。

(2)后缀表达式的当前元素与后续元素的比较。这种情况的可能性比较小，因为当前元素影响写入的情况也不合乎逻辑。

(3)中缀表达式的当前元素与栈顶元素比较。这种情况的可能性比较大。

(4)后缀表达式的当前元素与栈顶元素比较。这种情况的可能性也比较大。

(5)栈顶元素与其下面在栈元素的比较。这种情况的可能性比较小，与栈内其他元素比较以决定是否出栈是不合乎逻辑的。

以上述分析为基础，在头脑中模拟程序的运行过程，看这样是否能够解决问题。通过这样的模拟运行，就能够对以上分析作更精确的判断，从而正确地解答试题。在解答了问题 1、问题 2 和问题 3 后，问题 4 的答案即可得出。详细的模拟运行过程和具体答案，这里应当再重复了。

如果说这里的解题过程有什么经验可循，就是透彻的理解和周密的思维，另外分析问题的速度也是一个重要方面。以上解析达 2000 多字，其实在考试中一定要在 30 分钟之内完成这些分析，否则就要占用其他试题的解题时间了。

【答案】

[问题 1] PRIOR(IN [i] :PRIOR(S [P])

［问题 2］

功能：将当前符号 IN［i］入栈。

操作：p+1→p

IN［i］→P［p］

［问题 3］

功能：出栈(将栈顶元素送往数组 POLISH［］)

操作：k+1→p

S［p］→POLISH［］

P-1→p

［问题 4］AB+CD*-EF-*G

试题 3 (2000 年试题 3)

阅读以下说明和流程图(如图 6-3～图 6-6 所示)，回答问题 1 和问题 2，将解答写在答卷的对应栏内。

［说明］

某供销系统接受顾客的订货单。当库存中某配件的数量小于订购量或库存量低于一定数量时，向供应商发出采货单；当某配件的库存量大于或等于订购量时，或者收到供应商的送货单并更新了库存后，向顾客发出提货单。该系统还可随时向总经理提供销售和库存情况表。该供销系统的分层数据流图中部分数据流和文件的组成如下：

文件

配件库存=配件号+配件名+规格+数量+允许的最低率库存量

数据流

订货单=配件号+配件名+规格+数量+顾客名+地址

提货单=订货单+金额

采货单=配件号+配件名+规格+数量+供应商名+地址

送货单=配件号+配件名+规格+数量+金额

假定顶层图是正确的，"供应商"文件已由其他系统生成。

［问题 1］

指出哪张图中的哪些文件可不必画出。

［问题 2］

指出哪些图中遗漏了哪些数据流。回答时用如下形式之一：

(1)××图中遗漏了××加工(或文件)流向××加工(或文件)的××数据流；

(2)××图中××加工遗漏了××输入(或输出)数据流。

【解析】

流程图试题的解答较其他类型的试题要直观一些，但要求考生必须特别仔细，百密一疏也会导致严重的失分。

首先阅读试题说明，明确供销系统的基本结构。基本结构如项层图(图 6-3)所示，供销系统与顾客、供应商和总经理之间交换数据。供销系统接受顾客的定货单，反馈不合法的定货单，并在适当的时机向顾客发出提贷单；供销系统向供应产发出采购单，接受送货单，并反馈不合法的送货单；在以上购销操作的基础上，该系统接受总经理的并返回销售及库存情况表。

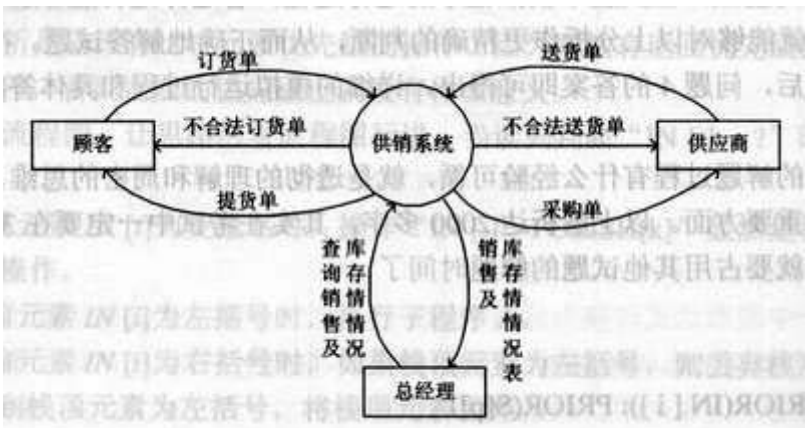


图 6-3 顶层图

问题 1 要求指出哪张图中的哪些文件是不必画出的。这是一个难度较高的要求。首先，它要求仔细研究每一张图，确认每张图中每个文件是否必须画出；其次所谓不必画出，其含义是该文件画出也不会导致该流程图发生错误，但不画出该文件将使整个流程图的层次结构更科学、更清晰。一般来说，如果一个元素(文件如数据流等)只用于一个加工，那么它就不必在其上层流程图中画出。但试题说

明中只假定了顶层图的正确性，对于其他流程图，我们首先要确定它们是否完整、是否正确。

对照试题说明，顶层图简单明了，甚至没有涉及文件，因此我们可以跳过顶层图不予考虑，直接研究 0 层图(图 6-4)。0 层图中涉及到 3 个文件，即配件库存、缺货订单和采购清单。这里缺货订单和配件库丰都在销售和采购两个加工中使用，只有采购清单只用于采购加工的似乎应该判断采购清单是不必画的，但由于试题不保证项层图以外各加工子图的正确性，难道就此确定在 0 层图采购清单文件不必画出的?不，在各个加工子图中，有可能发生文件、数据流、加工的遗漏或错画，影响我们在此处的判断。因此，只有通过仔细研究以上 3 个文件在各个加工子图中应用印证我们的判断。

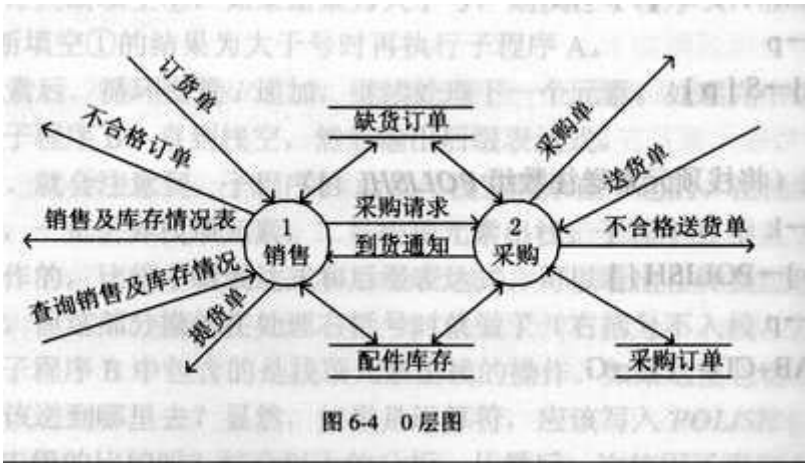


图 6-4 0 层图

首先看配件库存文件。在加工 1 子图(图 6-5)中，配件库存文件经过加工 1.4 更新库存而修改，然后还要为加工 1.1 和加工 1.2 提供数据支持，在加工 1.1 中，通过检查订货单的订购的配件在配件库存中是否有记录来确定该配件是否属于经销范围，然后以此来确定订货单是否合法。其次，还要在加工 1.2 中根据比较合法订货单的数量与库存数量，以此确定是否需要发出采购请求。再交，在发出提货单后，如果库存量低于允许的最低库存量，也要发出采购注。可以看出，在该加工图中，未发现对配件库存文件的不适当的使用。在加工 2 子图(图 6-6)中，在加工 2.4 核对送货单后要更新库存文件然后加工 2.3 计算增量提供数据支持，计算出需要采购的配件的数量。应该说这里对配件库存 文件的使用了是正确的。因此我们可以确定配件库存文件在加工 1 和加工 2 中都有应用，画在 0 层图中是合适的。

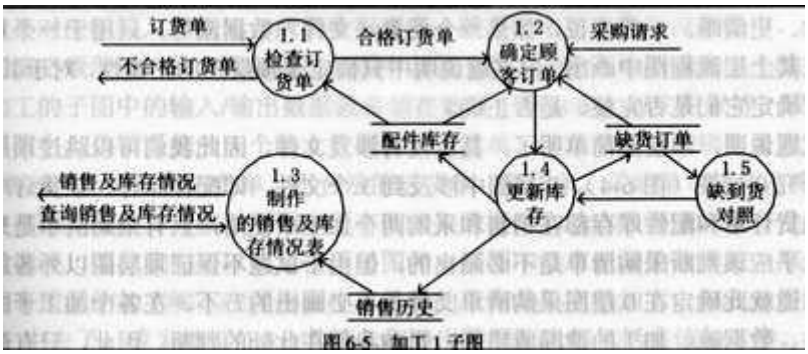


图 6-5 加工 1 子图

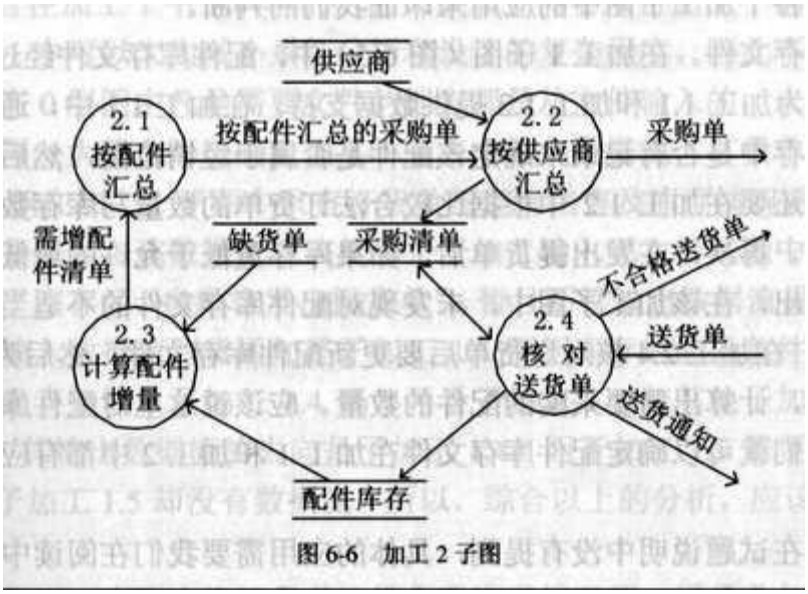


图 6-6 加工 2 子图

缺货订单文件在试题说明中没有提到，具体的应用需要我们的阅读中体会。0 层图中显示该文件被加工 1 和加工 2 使用，且在两个加工中也都有所有应用。现在的任务就是确定两个加工中对该文件的使用是正确的。在加工 1 中缺货订单的数据来自加工 1.2 和加工 1.4，在加工 1.2 中，接受合法的订货单后，如果订货单上数量大于配件库存文件的数量，将产生缺货订单文件。在加工 1.4 中，在有货订单送达后，需要交出提货单，然后更新库存如果库存量低于允许的最低库存量，应该将缺货信息反映到缺货订单中。在加工 2 中，缺货订单为加工 2.3 提供部分数据支持，在计算配件增理以明确需要增加的配件的清单时，需要参照缺货订单。结合以上对使用缺货订单文件的分析，可以认为该文件在加工 1 和加工 2 中的应用都是正确的，因此将该文件画在 0 层图中是有必要的。

再来看采购清单。0 层图显示该文件只应用于加工 2。采购清单如果就应用在加工 1 中，最有可能的地方是加工 1.2 及其后的采购请求数据流，但在这里采购清单是有必要的吗?从加工 2 中可以看出，采购清单是按供应商对采购单进行汇总来产生的，对采购单进行汇总，显然已经属于加工 2 的工作，不应在加工 1 中予以反映。由此可以断定在加工 1 中不应该有采购单文件。所以在 0 层图中画采购清单是不合适的。

以上考察了 0 层图、加工 1 子图和加工 2 子图中缺货订单、配件库存及采购清单的使用，下面研究加工 1 中的销售历史文件。如果没有该文件支持，加工 1.3 是无法制作销售及库存情况表的；而且该文件由加工 1.4 更新库存产生，其输入数据流和输出数据流均在加工 1 中，在加工 1 中使用该文件是合适的。

问题 2 要求指出哪些图中遗漏了哪些元素。一般来说，这类题目的解答首先要考虑各层次图的数据平衡，其次要考虑加工的输入数据流和输出数据流要平衡，即保证加工的输出数据流都有其对应的输入的数据流。所谓数据平衡，就是在多层次数据流程图中，父图和子图之宰的数据流必须保持一致，比如说在父图中某加工有 2 个输入数据流和一个输出数据流，那么在该加工的子图中的输入/输出数据流必须在数目上和内容上与父图保持一致。

在顶层图中，供销系统的 3 个输入数据流(订货单、送货单与销售及库存情况)和 5 个输出数据流(不合法的订货单、不合法的送货单、提货单、采购单与销售及库存情况表)在 0 层图中都得到反映。考察 0 层图中所有的输入数据流和输出数据流，没有发现超顶层图中是否会有遗漏的加工呢?在顶层图中总经理的查询是单立的加工，但在 0 层图中却给包括到加工 1 中去了，成为加工 1.3，其输入或输出数据流也都包括在该加工中。考察加工 1 子图，加工 1.3 包含在加工 1 中是科学的，因为该加工需要来自加工 1 数据支持，且事务简单，包含在加工 1 中可以大大减小系统分析和设计的复杂程度。因此不能说 0 层图遗漏了加工。至于文件，在 0 层图中凡是需要文件的地方都是从文件输入的，未发生遗漏现象。

仔细研究 0 层图，大体上确定加工 1 有 3 个输入数据流(订货单、到货通知和查询销售及库存情况)，便在加工 1 子图中只有订货单与查询销售及库存情况两个数据流，显然遗漏了输入数据到货通知，该数据流应该从哪个子加工输入呢?看试题中的说明"收到供应商的送货单并更新了库存后，向顾客发出了提货单"。暂且不管输出数据流提货单，根据试题说明，至少可以判定输入数据流到货通知是子加工 1.4 更新库存的前提条件，没有到货通知，就无法更新库存。是直接输入加工 1.4 吗?再看子加工 1.4 与其他子加工的关系。子加工 1.5 是缺到货对照，其输出数据流的去向是子加工 1.4。而子加工 1.5 需要到货通知的支持，但从子加工 1.4 到子加工 1.5 却没有数据流。所以，综合以上分析，应该在子加工 1.5 处输入数据流到货通知。

考察 0 层图，加工 1 有不合格订货单、销售及库存情况表、提货单与采购请求 4 个输出数据流。但在加工 1 子图中，却不有输出数据流提货单，这显然就不符合数据平衡原则的。但该数据流应从何处输出呢?仍旧是上面引用的试题说明，在更新库存后向顾客发出提货单。显然这里合适的位置是子加工 1.4。

在 0 层图中加工 2 有 3 个输出数据流，分别是采购单、不合格送货单与到货通知。在加工 2 子图中这 3 个输出数据流都有反映，因此可以确定在加工 2 子图中没有遗漏输出数据流。在 0 层中，加工 2 有两个输入数据流(送货单与采购请求)，显然加工 2 子图中遗漏了输入数据流采购请求，问题是该数据流从哪个子加工输入。

子加工 2.4 的任务是核对送货单，显然采购请求不应该在这里输入；那么在子加工 2.1、子加工 2.2 和子加工 2.3 中，究竟应该在何处输入采购请求数据流呢？如果在子加工 2.2 中输入，则由子加工 2.1 到子加工 2.2 数据流按配件清单必将是不完全的，因为这样只计算了缺货订单上需要的配件数量。所以，只能从子加工 2.3 输入采购请求。

由于上面已经确认 0 层图没有遗漏的数据流，我们就不必担心在 0 层图中遗漏了加工 1 和加工 2 中的某个数据流。

接着看加工 1 子图和加工 2 子图中的各个子加工，看它们的输入数据流和输出数据流是否平衡。子加工 1.1、1.2、1.4、1.5 和子加工 2.1、2.2、2.3、2.4 都没有问题，只有子加工 1.3，其输出是销售及库存情况表，但在图中该子加工只从文件销售历史中输入数据，销售历史中显然不包含反映库存情况的数据，因此我们可以断定，加工 1 子图中遗漏了从文件配件库存到子加工 1.3 的数据流。

最后再仔细研究整个数据流程图，未发现数据不平衡现象。
【答案】

［问题 1］

0 层中的"采购清单"不必画出。

［问题 2］

加工 1 子图中遗漏了"配件库存"文件到 1.3 加工的数据流。

加工 1 子图中 1.4 加工遗漏了"提货单"输出数据流。

加工 1 子图中 1.5 加工遗漏了"到货通知"输入数据流。

加工 2 子图中 2.3 加工遗漏了"采购请求"输入数据流。

6.2 1999 年度软件设计试题解析

1999 年度下午试卷共有 3 道软件设计试题，选答 2 道，比较容易。试题 1 的 3 个问题是历年试题中比较常见的类型 ，没有出现描述某个处理流程、修改流程图以适应新的要求等令人生畏的问题。与历年试题比较，由于问题的规模较小，试题 2 也相对容易。试题 3 给出了一个翻译流程的状态转换矩阵和语义动作矩阵，要求考生根据句法图、流程图和矩阵中已有内容填写两个矩阵。这种试题的解答的关键是要在理解句法图和语法图的基础上理解各种状态的含义，由此可以确定各种状态下相应的语义动作。在理解过程中，要注意利用矩阵中已有部分来推测未知部分。
试题 1 （1999 年试题 1）

阅读以下说明和流程图 6-7，回答问题 1 至问题 3，将解答写在答卷的对应栏内。

［说明］

本流程图描述了某仓库物品入出库管理的处理流程。每张入库单都由两们操作员分别录入，经处理 1 或处理 3 输入系统合作性检查，并将合法的入库单或出库单记入入库单文件或出库单文件。然后通过处理 2 或处理 4 实时更新库存文件。处理 5 每周执行一次，它依次检查库中的每一种物品，当某物品的库存小于该物品的最低库存量时，制订采购计划，输出订购单。处理 6 和处理 7 每月执行一次，处理 6 将入库单文件和出库单文件合并成月入库文件，并根据统计的要求对其进行排序。处理 7 进行统计，产生月报表，并把该月合并后的月入出库文件添加到月入出库后备文件中，以备日后查找。最后清除入库单文件、出库单文件和月入出库文件。

系统中某些文件和报表的格式如下：

库存文件记录。物品编号+名称+规格+库存量+最低库存量+最高库存量(其中"最高库存量"指该物品允许存放在库中的最大值。)

入库单文件记录。日期+物品编号+数量

出库单文件记录。日期+物品编号+数量

月报表格式

物品编号	日期	入库数	出库数
×××× ××	×× ×× ××	×× ×× ×× ……	×× ×× ××
	当月小计	×××	×××
××××	×× ××	×× ×× ……	×× ××

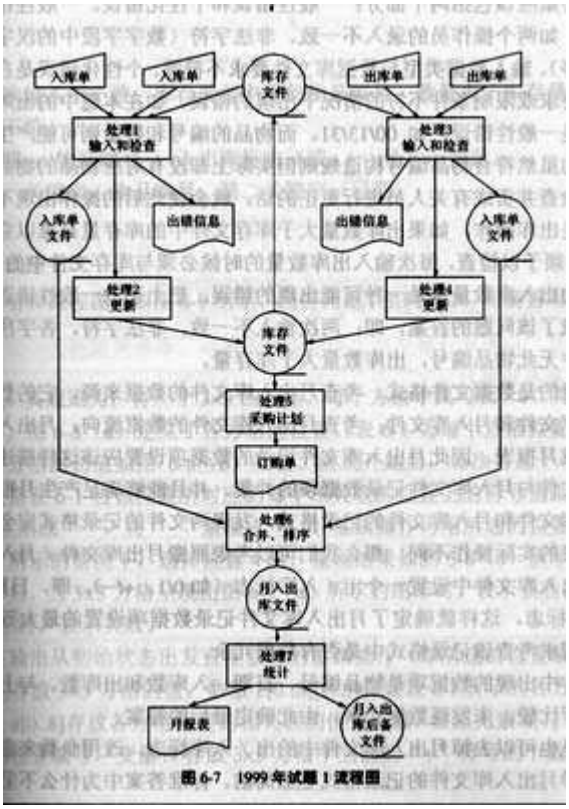


图 6-7 1999 年试题 1 流程图

［问题 1］

指出处理 3 能检查出库单中的哪些错误。

［问题 2］

指出月入出库文件的记录格式。

［问题 3］

指出处理 6 排序的第一和第二关键字。

【解析】
该题涉及的是某仓库的物口出入管理系统的处理流程图。仔细阅读试题说明和流程图，会发现需要特别注意处理 3 和处理 6

问题 1 涉及的是数据输入与合法性检查，这里需要回答在处理 3 中能够检查出哪些错误。关于数据合法检查的问题，几乎在历年的软件分析试题中都会出现。解答此类题目也有一定的套路，即答案应该包括两个部分：一般性错误和个性化错误。一般性错误即通常都会出现的录入错误，如两个操作员的录入不一致、非法字符(数字字段中的汉字字符、非数学标记符号的字符等)、输入数据类型与数据库文件要求不符等。个性化错误是在输入过程中与具体的数据文件要求或限制文件不符的情况下出现的错误，如在本题中的出库单文件中关于日期可能产生的是一般性的错误，如 00/13/31。而物品的编号和数量则可能产生个性化错误，如

由于笔误造成的虽然符合物品编号构造规则但实际上却没有对应物品的物品编号。如果不在处理 3 中予以检查并责成有关人中员进行更正的话，就会使此后的操作出现不可预测的错误。另外，由于这是出库操作，如果出库数量大于库存文件中的库存量是难以完成一次出库操作的，在处理中必须予以检查，每次输入出库数量的时候必须与库存文件中的库存量进行比较。同时，非正数的出入库数量也是一种可能出现的错误。把上述的一般性错误和个性化错误合并在一起就形成了该问题的答案，即：两次输入不一致，非法字符，各字段的数据类型不一致，库存文件中无此物品编号，出库数量大于库存量。

问题 2 涉及的是数据文件格式。考查月出入库文件的数据来源，它的数据来源是经过合并排序的月出库文件和月入库文件；考查月出入库文件的数据流向，月出入库文件经过处理 7 的统计后形成月报表。因此月出入库文件记录的数据项设置应该这样描述：其最大范围不应超出月出库文件与月入库文件记录数据项的并集，并且能够满足产生月报表的数据需求。

考查月出库条件和月入库条件的记录格式，发现两文件的记录格式完全相同，所不同的是每条记录代表的实际操作不同。那么我们考虑照月出库文件/月入库条件的记录格式，然后在月出入库条件中设置一个出/入库标志(如 0/1，+/-)，即，日期+物品编号+数量+出入库标志，这样就确定了月出入库文件记录数据项设置的最大范围。再根据月报表文件的数据需求考查该记录格式是否数据冗余。

月报表文件中出现的数据项是物品编号、日期、入库数和出库数。与上述的月出入库记录格式进行比较，未发现冗余，由此确定最后答案。

其实在这里也可以去掉月出入库文件中的出/入库标志，改用数来表示出库，正数来表示入库，这样月出入库文件的记录格式更加简洁。标准答案中为什么不采用这种方案，请读者自己思考。

其实问题 2 的解答相当容易，在这里用这么多文字进行解析，关键是要说明解答此类试题的科学思维方示，即通过数据来源确定记录中数据设置的最大集合，通过数据流出方向消除其数据冗余。对于复杂的试题来说，难度主要集中在消除数据冗余这一部分，希望读者在复习备考过程中加强这方面的练习。

问题 3 考查的排序关键项问题。首先应该明确，排序的目的是为了便此后的处理，提高系统的效率，但只有选择正确的排序关键项才能达此目的，否则只能起反作用。这也是解答此类题目的关键所在。

处理 6 中的排序产生月出入库文件。如果不继续往下看，我们能够意识到可以以日期和物品意识到可以以日期和物品编号为排序关键项，便没有更多的证据来确谁是第一关键项，谁是第二关键项。

考查处理 7，处理 7 使用的月出入库文件。如果不继续往下看，我们能够意识到可以以日期为物品编号为排序关键项，但没有更多的证据来确定谁是第一关键项，谁是第二关键项。
考查处理 7，处理 7 使用的月出入库文件就是处理 6 合并排序的结果，那么关键

项的高以置必须有利于处理 7，而处理 7 产生的两个结果中，从月报表格式可以明显看出是先以物品编号为主，然后再按日期罗列出/入库操作。这样，我们就可以确定，如果处理 6 中的排序不把物品编号作为第一关键项的话就是一种失败，产生月报表时的效率必然很低。确定了第一关键项，第二关键项也就很明显了。所以，在处理 6 中必须以物品编号为第一关键项，以日期为第二关键项。

【答案】

［问题 1］两次输入不一致；库存文件无此物品编号；数量大于库存量；非法字符；各字估的数据类型不符。

［问题 2］日期+物品编号+入出库标志(或+/-号)+数量

［问题 3］第一关键字：物品编号；第二关键字：日期。

试题 2（1999 年试题 2）

阅读以下说明和流程图，如图 6-9 所示，回答问题 1 至问题 3，将解答写在答卷的对应栏内。

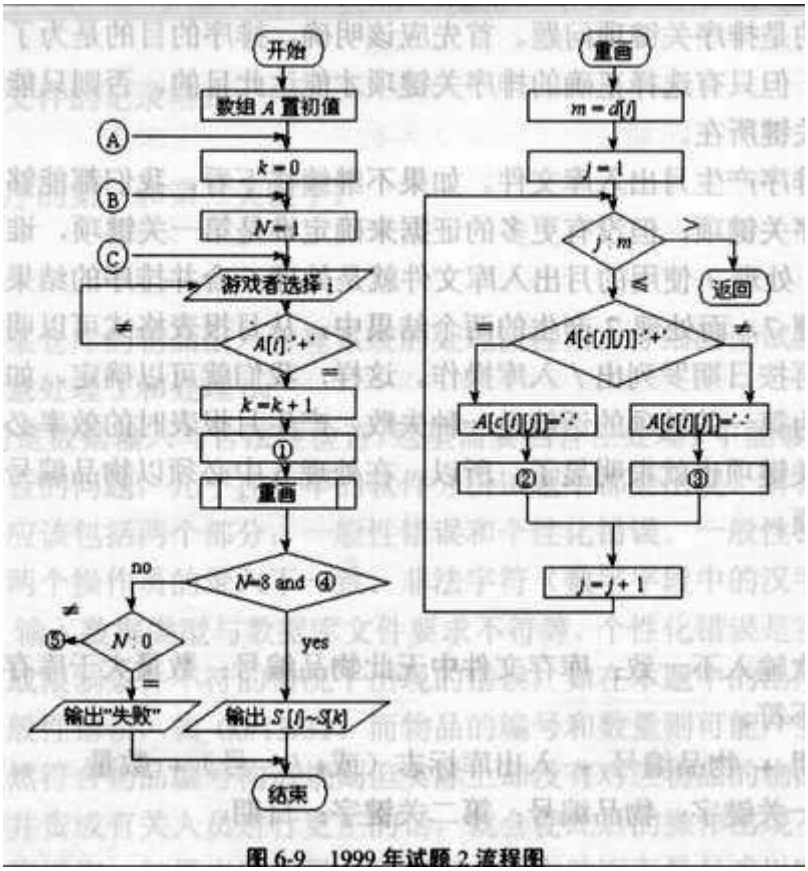


图 6-9 1999 年试题 2 流程图

［说明］

有一种游戏，其规则为：有一个 3×3 的方格，每个方格中只可画"+"符号或"-"符号，表示该方格的值。图 6-8(a)定义了各方格的位置，表 6-1 为每个方格位置定义了与其相关联的位置集，各方格的初值如图 6-8(b)所示。游戏开始后，每次可选一个值为"+"的方格位置，然后根据表 6-1 将该位置所对应的和线个相关联的位置上的符号重画成其不同的符号即将"+"重画成"-"，将"-"重画成"+"。重画操作可用所选位置编号来描述。例如如图 6-8(b)所示的情况下，选择位置 4 时，重画结果如图 6-8(c)所示，经过边续的若干次操作后，当 3×3 方格呈现出图 6-8(d)所示的图形时，表示获胜；当呈现出图 6-8(e)所示的图形表示失败。

表 6-1 方格位置及其相关位置集的对照表

方格位置	相关位置集
0	0134
1	012
2	1245
3	036
4	13457
5	258
6	3467
7	678
8	4578

流程图旨在输出从初始状态出发直至获胜的重画操作(即所选的位置编号)序列。图中假定数组 A［0..8］存放 3×3 方格值，数组 c〔0..8〕〔1..5〕存放表 6-1 所示的各方格位置的相关联的位置集，数组 d〔0..8〕存放各方格位置的相关联的位置个数，数组元素 S〔1〕~S〔k〕存放各次重画操作所对应的位置编号，变量 N 存放 3×3 方格中当前的"+"符号的个数。

012	----	-+-	+++	---
345	-+-	+-+	+-+	---
678	---	-+-	+++	---
(a)	(b)	(c)	(d)	(e)

〔问题 1〕 填充图中的①~④。

〔问题 2〕 图中的⑤应与 A、B 和 C 中的哪一点连接。

〔问题 3〕 如果每次由游戏者选择方格改由程序自动枚举选择，那么，为从初态出发求出所有可能的获胜重画操作序列，在哪些情况下需要进行回溯处理。

【解析】

这是一道典型的流程图试题，主要考查考生完善和评价流程图的能力，解答此类试题的常用方法是：先弄清楚流程图中各个变量 的含义以及处理过程中各种状态的标志，再分析流程图的宏观结构，然后结合试题说明理解算法说明在流程图中是同哪部分流程图来实现的

首先，通过阅读试题说明，我们将各个变量的含义罗列出来：

(1)数组 A〔0..8〕存放 3×3 方格的值，其值为"+"或者"-"；

(2)数组 c〔0..8〕〔1..5〕存放各方格位置的相关联的位置集；

(3)数组 d〔0..8〕存放各方格位置的相关联的位置个数；

(4)数组元素 S〔1〕~S〔k〕存放各次重画操作对应的位置编号；

(5)变量 N 存放 9 个方格中当前"+"符号的个数。

关于流程图中各个变量的含义，我们结合对流程图的理解来明确其含义。同时，仔细观察成功时各方格的值，并结合变量设置及流程图中关于成功/失败的判断，可以推测流程图中以"+"个数为 8 且位置 4 的方格值为"-"来判断成功，而以"+"个数为 0 来判定失败。

现在让我们带着问题 1 和问题 2 来理解流程图。首先，直到语句框 N＝1，都是处理流程图中的赋初值操作，N＝1 是很好理解的，因为在方格的初值中只有一个"+"，但这里的 k=0 是什么意思?往后看，语句框 k=k+1 没有提供多少关于 k 的有效信息，而输出 S〔1〕~S〔k〕却让我们意识到 k 标志序列中已加入方格的个数，同时，S〔k〕也就表示第 k 个被选中方格的位置。在判断每次选择的都是一个值为"+"的方格之后，流程图进入实际处理过程。而 i 也就是当前被选中的方格的位置，那么在重画部分的 d〔i〕也就是当前被选中方格的相关位置的个数，于是 A〔c〔i〕〔j〕〕也就是当前被选中方格的第 j 个相关位置的值了。

基于以上的理解，由于 S〔k〕存储的第 k 个被选中方格的位置，因此在选定了方格后应该将其位置记入其位置记入对应的数组元素中。考虑到语句框①之后的操作是重画，这里应该完成上述操作因为这是一个循环结构(虽然我们还没有确定填空⑤到底连接到哪里)，并且考虑到 k 和 i 在流程图中的含义，所以我们认为这里应当是 S〔k〕=i。

现在可以跳过重画部分不管，解决填空④和⑤。在经过④的判断后，在 yes 分支中处理了成功怀况下的输出位置序列操作，那么这里的判断应该是判断是否成功。结合上我们成功时方格值分布情况的理解，在填空④中填写 A〔4〕='-'。

那么填空⑤呢?⑤虽然指示了在既未成功又未失败的情况下跳转的目的地，由此构成了一个完整的循环，也就是继续选择。

显然，跳转到 A、B 两处都不能实现继续选择的功能。跳转 A 处几乎是一切从头重来，但数组 A 的值是不确定的，而且变量 N 也不能正确反映"+"的个数，每次循环后总是只有一个"+"，而实际上不应是这样的，除非碰巧在第一次成功，否则很难成功。跳转到 B 处，则可以保证已经选择了方格的个数是正确的，但变量

N 仍不能正确反映"+"的个数。由此可见，跳转 A、B 两处都是不正确的，只能跳转到 C 处。

现在我们来看重画部分的流程图。结合对变量的理解，我们可以认定，填空②和③是对一个相关位置的方格时行重画以后，又进行某些操作。每次重画都涉及"+"数量的增加或减少，这也是在流程图中必须处理的。综合考虑流程图的结构，应在②和③中分别填写 N=N-1 和 N=N+1 来完成上述处理。至此问题 1 和问题 2 都解决了。

问题 3 是关于在使用程序自动枚举选择的情况下何时进行回溯处理的问题。首先，在一次成功或失败时需要回溯处理，这是很明显的；其次，由于每次都必须选定一个值为"+"的方格，如果出现了 9 个方格中只有一个"+"，必须导致失败的情况，那么再次出现这种情况 时应该自动回溯处理，可能的情况如下所示。
另外也不能忽视，在一种枚举选择完毕后必须进行回溯处理，否则就不能够实现穷尽所有成功情况的重画操作。

012	+--	-+-	--+	---	---	---	---	---	---
345	---	---	---	+--	-+-	--+	---	---	---
678	---	---	---	---	---	---	+--	-+-	--+

【答案】

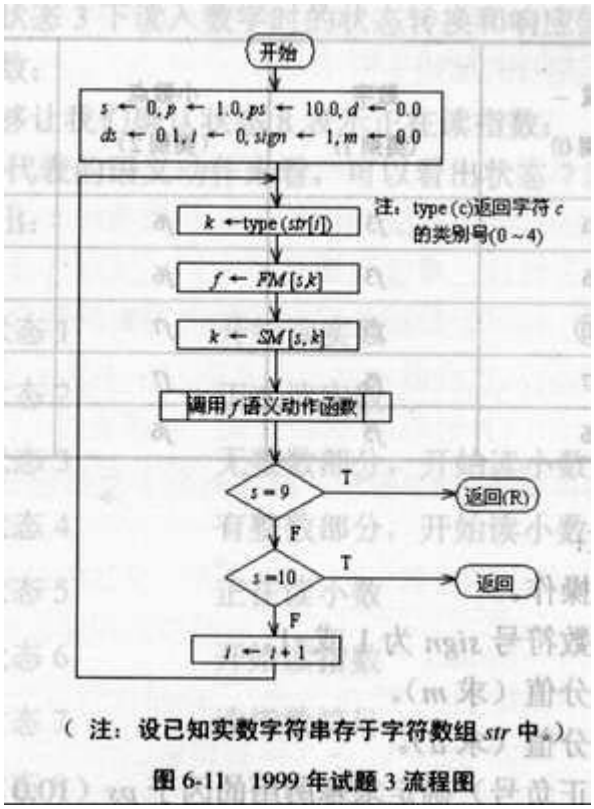
[问题 1] ①S〔k〕=i②N=N-1③N=N+1④A〔4〕='-'

[问题 2] ⑤C 相连。

[问题 3] ①获胜。②失败。③出现了以前出现过的图形。④一种图形的枚举选择完。

试题 3 (1999 年试题 3)

阅读以下说明和流程图 6-11,回答问题 1 和问题 2,将解答写在答卷的对应栏内。



（注：设已知实数字符串存于字符数组 *str* 中。）

图 6-11 1999 年试题 3 流程图

［说明］

本流程图采用状态矩阵方法将已知字符序列翻译成实数(其句法图如图 6-10 所示)。本题的状态矩阵分成两部分，状态转换矩阵 SM(见表 6-2)和语义动作矩阵 FM(见表 6-3)，它们分别存放每个状态遇到某字符时应执行的语义动作以及执行动作后应转移到的新状态。

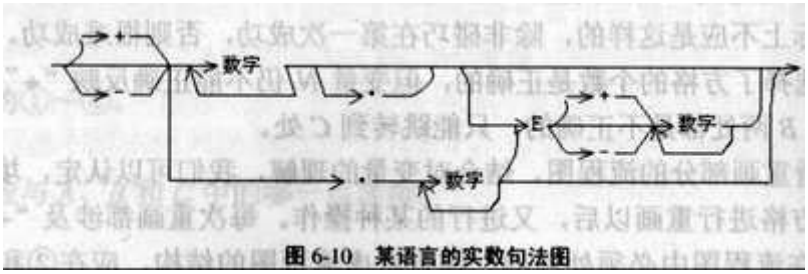


图 6-10 某语言的实数句法图

表 6-2 状态转换矩阵 SM

状态 \ 字符类	+ 或 -（类别 0）	数字（类别 1）	小数点（类别 2）	字符 E（类别 3）	其他字符（类别 4）
0	1	2	3	10	10
1	10	①	②	10	10
2	9	2	4	6	9
3	10	5	10	10	10
4	9	③	9	④	9
5	9	5	9	6	9
6	7	8	10	10	10
7	10	⑤	10	10	10
8	9	8	9	9	9

表 6-3 语义动作矩阵 FM

状态 \ 字符类	+ 或 -（类别 0）	数字（类别 1）	小数点（类别 2）	字符 E（类别 3）	其他字符（类别 4）
0	f1	f2	f0	f7	f7
1	f7	⑥	f0	f7	f7
2	f6	⑦	f0	f0	f6
3	f7	⑧	f7	f7	f7

续表

状态 \ 字符类	+ 或 -（类别 0）	数字（类别 1）	小数点（类别 2）	字符 E（类别 3）	其他字符（类别 4）
4	f6	f7	f6	f0	f6
5	f6	f7	f6	⑨	f6
6	⑩	f6	f7	f7	f7
7	f7	f6	f7	f7	f7
8	f6	f6	f6	f6	f6

本流程图从 0 状态出发逐个读入字符，在执行了 FM 中相应的语义动作后，转移到 SM 中指出相应新状态，重复这一过程，直至到达 9 状态或 10 状态。9 状态表示已正确地把该字符序列翻译成实数(注意，此时已多读进实数后的下一个字符)，10 状态表示出错。

问题 3 考查的是关键项问题，首先应该明确，排序的目的是为了便于此后的处理，提高系统的效率，但只有选择正确的排序关键项才能达到此目的，否则只能起反作用。这也是解答此类题目的关键所在。

现有以下语义动作函数：

f0，空函数，不做任何操作。

f1，按当前字符确定实数符号 sign 为 1 或-1。

f2，翻译实数的整数部分值(求 m)。

f3，翻译实数的小数部分值(求 d)。

f4，按当前字符(幂的正负号)确定求幂所用的因子 ps(10.0 或 0.1)

f5，翻译实数的幂(求 p)

f6，求实数的值 R=sign*(m+d)*p。

f7，输出错误信息。

［问题 1］将状态转换矩阵 SM 中①～⑤处的正确内容填入答卷的对应栏内。

［问题 2］将语义动作矩阵 FM 中⑥～⑩处的正确内容填入答卷的对应栏内。

【解析】

本题的解答关键是确定各种状态的含义，而理解这些状态的可靠信息源则是句法图、状态转换矩阵 SM 和语义动作矩阵 FM。

首先，考查在状态 0 下遇到各种情况时的状态转换及相应的语义动作 。在状态 0 下，如果读入一个"+"或者"-"，要转换为状态 1，并且执行语义动作 f1，按当前字符确定实数符合号 singn 为 1 或者-1；如果读入一个数字，则要转换为状态 2，并且执行语义动作 f2，翻译实数的整数部分值；如果读入一个小数点，则要转移为状态 3，执行语义动作 f0，即不做任何操作；在读入字符 E(指数标志)和其他字符时出错。

如果在头脑中模拟一台计算机来读实数的话，不难判断：

状态 1 表示开始读一个实数；

状态 2 表示正在读一个实数；

状态 3 表示无整数部分，开始读小数。

以同样的方法考虑在状态 2 下读入数字、读入小数点、读入 E 时的状态转换和相应的语义动作，能够得到以下判断：

状态 2 表示正在读一个实数，这在上面已经得出同样的结论；

状态 4 表示在有整数部分的情况下，开始读小数；

状态 6 表示开始读指数。

以同样的方法考虑在状态 3 下读入数字时的状态转换和响应的语义动作，可以认定：

状态 5 表示正在读小数；

状态 6 时读入数字能够让我们确认状态 8 表示正在读指数；

从状态 7 的位置及 f4 代表的语义动作来看，可以看出状态 7 表示指数符号。

把各种状态的含义列出：

状态 1 开始读实数

状态 2 正在读实数

状态 3 无整数部分，开始读小数

状态 4 有整数部分，开始读小数

状态 5 正在读小数

状态 6 开始读指数

状态 7 读指数符号

状态 8 正在读指数

基于以上分析，可以进行试题的解答。

①是在状态 1 下读入数字时的状态转换，自然是转换为状态 2，表示正在读实数；

②是在状态 1 下读入小数点，自然是转换为状态 3，表示无实数部分，开始读小数；

③是在状态 4 下读入数字，自然是转换为状态 5，表示正在读小数；

④是在状态 4 下读入 E，自然是转换为状态 6，表示开始读指数；

⑤是在状态 7 下读入数字，自然是转换为状态 8，表示正在读指数；

⑥是在状态 1 下读入数字需要执行的语义动作，我们可以确定这里需要填写 f2，即翻译实数的整数部分。

⑦是在状态 2 下读入数字，需要执行的语义动仍旧应该 f2；

⑧是在状态 3 下读入数字，其相应的语义动作就应该是 f3，即翻译实数的小数部分值；

⑨是在状态 5 下读入 E 时的相应语义动作，参考状态 2 时读入 E 的语义动作，这里填写 f0；⑩是在状态 6 下读入"+"或者"-"时的相应的语义动作，理解 f4 所代表的语义动作，这里必须填写 f4。

【答案】

[问题 1] ①2 ②3 ③5 ④6 ⑤8

[问题 2] ⑥f2 ⑦f2 ⑧f3 ⑨f0 ⑩f4

6.3 1998 年度软件设计试题解析

共 3 道软件设计试题，试题 1 考查数据处理流程，是一道软件分析试题。试题 2 考察软件测试，由于其内容的特殊性(知识内容集中出现)，只要考生对覆盖标准的理解正确，通过建立真值表或其他方法来解答题目，难度应该是不大的。试题 3 是流程图填空及修改，考查的是字符串处理方面的问题，相对于数组处理等其他内容来说直观性和可操作性强。

试题 1(1998 年试题 1)

阅读以下说明和流程图，如图 6-12 所示，回答问题 1 至问题 3，将解答写在答卷的对应栏内。

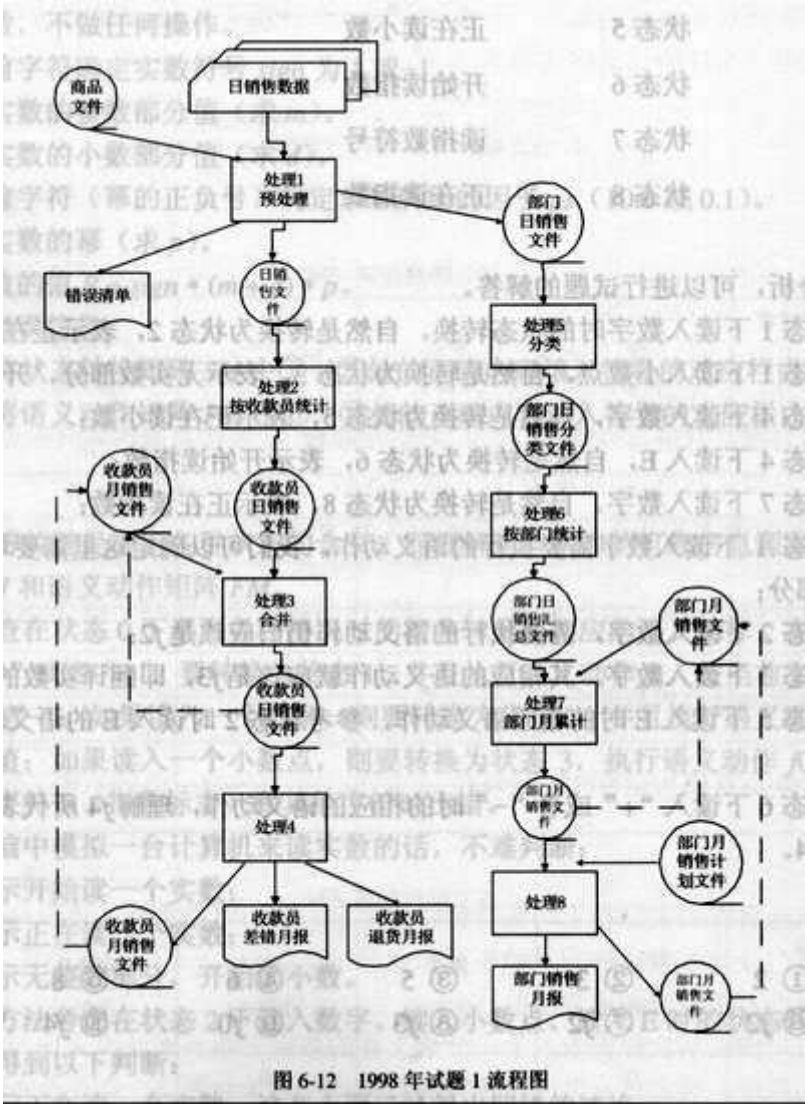


图 6-12 1998 年试题 1 流程图

［说明］

本流程图描述某超市销售数据的部分处理流程。超市中有若干台收款机和若干名收款员。这里，把一个收款员开始使用一台收款机到离开这台收款机称为该书款员的一次作业。作业开始时，收款员先在收款机上输入收款员和作业前金额。作业前金额是为了销售时的找零而在作业前预先放入钱箱的金额数。作业结束时，收款员要打开钱柜，取走全部现金，并把这些现金的金额数(称为作业后金额)输入收款机。

当作业前金额+本次作业售货总金额-本次作业退货总金额≠作业后金额时，表示这次作业存在金额差错。

本流程图已简化，并作以下假定：该超市只有现金交易(不用信用卡和礼券)；一个收款员因某种原因(如吃饭)在一天中可以有多个作业；销售方式只有售货和退货两种。

整个超市分成若干部门（如食品部、服装部），系统按部门统计一个月中各类货物的销售数量和金额，最后根据月销售计划文件分析各部门完成销售计划的情况。系统还统计每个收款员的差错情况和退货情况。

图中处理 4 和处理 8 每月的最后一天执行一次(营业结束后)，其他处理每天执行一次。

图中部分数据、文件的记录格式如下：

日销售文件记录，

(作业开始标识+收款机号+收款员号+作业前金额)

|(售货标记|退货标记)+货号+数量+金额)

|(作业结束标记+收款机号+收款员号+作业后金额)

日销售数据，收款机号+收款员号+作业前金额{(售货标记|退货标记)+货号+数量+单价+金额}+作业后金额

部门日销售文件记录，部门号+(售货标记|退货标记)+货号+数量+金额

部门月销售计划文件记录，部门号+月计划金额

收款员差错月报，月份+收款员号+差错作业数+差错总金额

收款员退货月报，月份+收款员号+退货次数+退货总金额

其中{w}表示 w 重复出现多次；a|b 表示 a 或 b；a+b 表示 a 与 b。

[问题 1]

分别写出收款员日销售文件、商品文件、部门日销售汇总文件至少应包含哪些数据项。

[问题 2]

处理 1 能检查出日销售数据中的哪些错误。

[问题 3]

处理 4 对收款员月销售文件作何种操作。

【解析】

软件分析试题解答的关键在于切实理解软件需求。我们可以看到，该系统要完成3项任务，即(1)按部门统计各类货物的月销售量和金额，(2)分析各部门完成销售计划情况，(3)统计每个收款员的差错情况和退货情况。

对于问题1，需要从软件需求入手，分析文件的数据来源和通过该文件与其他文件共同作用需要产生的数据，确定该文件应该包含的数据项；对于问题2，要考查处理1所处理的数据对象，然后分析该数据对象可能出现的错误；对于问题3，关键是要看处理4的任务和处理前后收款员月销售文件数据的作用。由于本题涉及面广，限于篇幅，本书只能结合3个问题的解答来解析本题。

可以看出，商品文件和日销售数据通过处理1生成错误清单、日销售文件和部门日销售文件。首先，商品文件应该记录有关商品的一些数据，应该有一个标识商品的记录主键，而在系统中又多处用到货号这个数据项作为处理的依据。因此，选用在商品文件中包含货号，并以之作为记录主键是合理的。在部门日销售文件中可以看到部门号这个数据项，但在日销售数据中却没有该数据项。很简单，在商品文件中应该有部门号这个数据项，否则部门日销售文件中的部门号就没有来源。包含了货号和部门号的商品文件应该基本可以满足处理1的数据需求，但这里包含一种潜在的危险性，即日销售数据中的价格存在错误。如果在商品文件中不包含一种潜在的危险性，即日销售数据中的价格存在错误。如果在商品文件中不包含一个关于价格的数据项，这种错误将无法排除，所以商品文件至少应该包含货号、部门号和价格
3个数据项。

处理1的处理对象就是商品文件和日销售数据。根据数据对象的特点来分析可能在以下几个方面出现错误。(1)在日销售数据中存在金额、作业前金额和作业后金额3个数据项，保证数据的完整性来考查，对于数字型的数据一般要检查其合法性，即该数据是否在允许的取值范围之内，如作业前金额和作业后金额都不可能是负数，金额的取值是固定的，即应该是数量与单价的乘积。(2)不排除日销售数据中存在非法字符的可能性，因为各种误操作都是可能的。(3)上面已经提到，在商品文件中包含一个价格的数据项，以便检查日销售数据中与货号的正确对应。可能产生进入日销售数据文件的错误也就是这些，通过以上分析不难得出可能发生的错误。处理1应该能够检查出所有的错误，所以正确的解答应该包含以上3个方面的内容。

继续向下阅读处理流程，发现收款员日销售文件和收款员月销售文件进行了合并，说明收款员月销售文件的数据来源于收款员销售文件进行了合并，说明收款员月销售文件的数据来源于收款员日销售文件，即收款员月销售文件包含的数据项在收款员日销售文件中也应该包含，而且我们看到收款员日销售文件只用到这一次，所以基本上可以判定它们所包含的数据项应该是相同的。基于以上的分析，我们通过分析处理4对收款员月销售文件进行处理所产生的数据，来分析收款员日销售文件应该包含的数据项。处理4产产收款员月销售文件、收款员差错月报和收款员退货月报。处理4的输入和输出中都包括收款员月销售文件，因此在进行分析时该文件可以忽略不计，把收款员差错月报和收款员退货月报进行合并，就可以得到收款员日销售文件的结构，但有些数据项应该改名称，如差错总金额和退货总金额在这里显然是不合适的，可以改为差错金额的退货金额。

处理 4 对收款员月销售文件进行何种操作?因为该处理每月末执行一次，收款员月销售文件在通过处理 4 产生收款员差错月报、收款员退货月服之后已完成了任务，应该清空数据准备在下一个月份里记录收款员销售情况了，所以，在处理 4 中，收款员差错月服和收款员退货月报产生之后，应该删除其中所有记录，即进行初始化。

现在本题只有部门日销售汇总文件应包含哪些数据项没有解决，但部门日销售汇总文件通过处理 7 之后产生的部门月销售文件包含哪些数据项没有明确提示，只有部门日销售文件和部门月销售计划文件可供参考。再回过头仔细阅读本系统需要完成的 3 项任务，需要按部门统计各类货物的月销售量和金额，还需要分析各部门完成销售计划情况。显然，在部门月销售文件中应该包含月销售金额这个数据项，以便与月销售计划中的月计划金额进行比较，该数据项必须从部门日销售汇总文件中累积得到。系统要利用部门月销售文件统计各类货物的月销售量和金额，显然这两项数据的来源也是部门日销售汇总文件，而作为对货物分类统计的依据，数据项货号是必不可少的。除此之外，作为部门日销售汇总文件，应该以部门号作为记录主键。以上从对系统的数据支持角度来分析部门日销售汇总文件应该包括的数据项， 如果提到的数据项没有合法来源,那么系统就无法工作。部门日销售汇总文件的数据来源于部门销售文件，这证明分析是符合系统需求的。

【答案】

［问题 1］

收款员日销售文件：（每项 1 分）

收款员号+差错作业数+差错金额+退货次数+退还金额

商品文件：（每项 1 分）

货号+部门号+单价

部门日销售汇总文件：（每项 0.5 分，去除累计和的小数分）

部门号+货号+销售数量+销售总金额

［问题 2］

①存在非法字

②数量×单价≠金额

③货号（或单价）与商品文件中的货号（或单价）不一致

［问题 3］删除其中的所有记录或初始化。

试题 2 （1998 年试题 2）

阅读以下说明和流程图 6-13，回答问题，将解答写在答卷的对应栏内。

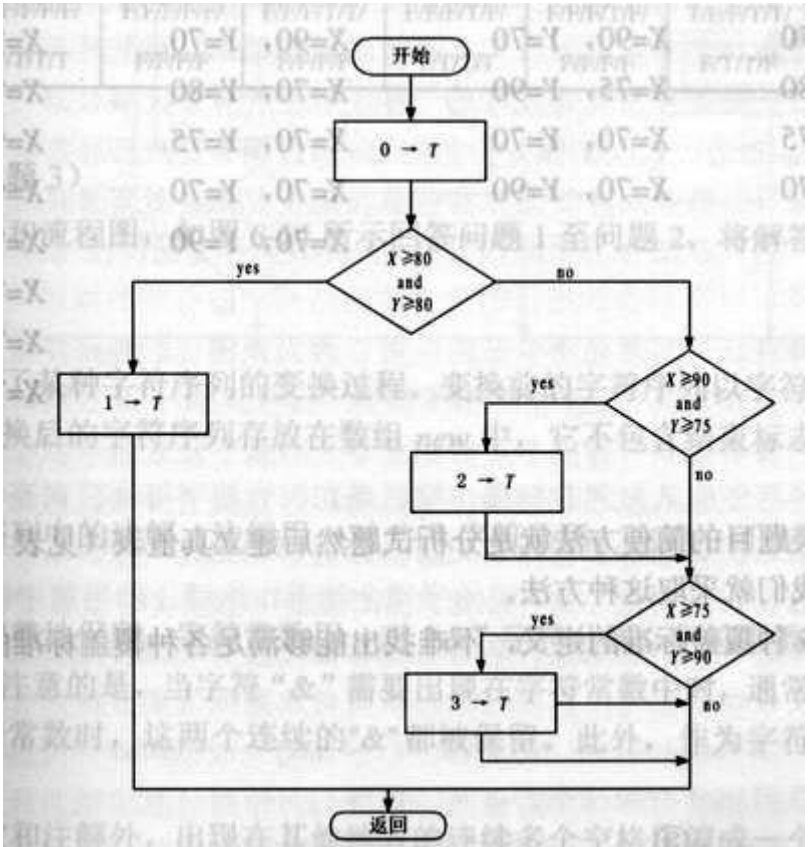


图 6-13 1998 年试题 2 流程图

［说明］

本流程图描述了某子程序处理流程，现要求用白盒测试法为孩子程设计测试数据。

［问题］

根据判定覆盖、条件覆盖、判定一条件覆盖、条件组合覆盖（即多重条件覆盖）和路径覆盖等 5 覆盖标准，从供选择的答案中分别找出满足相应覆盖标准的最小测试数据组（用①～ ⑿回答）。

供选择的答案

①X＝90，Y＝90②X＝90，Y＝70 ③X＝90，Y＝90④X＝90，Y＝75
X＝70，Y＝70 X＝70 Y＝90 X＝90，Y＝75 X＝75，Y＝90
X＝75，Y＝90 X＝70，Y＝70
⑤X＝90，Y＝90⑥X＝80，Y＝80 ⑦X＝80，Y＝80⑧X＝80，Y＝80
X＝90，Y＝75 X＝90，Y＝70 X＝90，Y＝75 X＝90，Y＝70
X＝75，Y＝90 X＝70，Y＝90 x＝90，Y＝90 X＝70，Y＝90
X＝70，Y＝70 X＝70，Y＝70 X＝75，Y＝90 X＝70，Y＝70

		X＝70，Y＝90	X＝75，Y＝75	
⑨X＝80，Y＝80	⑩X＝90，Y＝90	⑪X＝80，Y＝80	⑫X＝80，Y＝80	
X＝90，Y＝75	X＝90，Y＝75	X＝90，Y＝75	X＝80，Y＝70	
X＝90，Y＝70	X＝90，Y＝70	X＝90，Y＝70	X＝70，Y＝80	
X＝70，Y＝80	X＝75，Y＝90	X＝70，Y＝80	X＝70，Y＝70	
X＝70，Y＝75	X＝70，Y＝70	X＝70，Y＝75	X＝90，Y＝75	
X＝70，Y＝70	X＝70，Y＝90	X＝70，Y＝70	X＝90，Y＝70	
			X＝70，Y＝90	
			X＝70，Y＝75	
			X＝75，Y＝90	
			X=75， Y=80	
			X＝70，Y＝90	

【解析】

解答软件测试类题目的简便方法就是分析试题然后建立真值表（见表 6-4），通过真值表来解答试题，一在我们就采取这种方法。

表 6-4 真值表

	X>=80	Y>=80	X>=80 and Y>=80	X>=90	Y>=75	X>=90 and Y>=75	X>=75	Y>=90	X>=75 and Y>=90
①	T/F	T/F	T/F	T/F	T/F	T/F	T/F	T/F	T/F
②	T/F	F/T	F/F	T/F	F/T	F/F	T/F	F/T	F/F
③	T/T/F	T/F/T	T/F/F	T/T/F	T/T/T	T/T/F	T/T/T	T/F/T	T/F/T
④	T/F/F	F/T/F	F/T/F	T/F/F	T/T/F	T/F/F	T/T/F	F/T/F	F/T/F
⑤	T/T/F/F	T/F/T/T	T/F/F/T	T/T/F/F	T/T/T/F	T/T/F/F	T/T/T/F	T/F/T/F	T/F/T/F
⑥	T/T/F/F	T/F/T/F	T/F/F/F	F/T/F/F	T/F/T/F	F/F/F/F	T/T/F/F	F/F/T/F	F/F/F/F
⑦	T/T/T/F/F	T/F/T/T/T	T/F/T/F/F	F/T/T/F/F	T/T/T/T/T	F/T/T/F/F	T/T/T/T/F	F/F/T/T/T	F/F/T/T/F
⑧	T/T/F/F/F	T/F/T/F/F	T/F/F/F/F	F/T/F/F/F	T/F/T/F/T	F/F/F/F/F	T/T/F/F/T	F/F/T/F/F	F/F/F/F/F
⑨	T/T/T/F/F/F	T/F/F/T/F/F	T/F/F/F/F/F	F/T/T/F/F/F	T/T/F/T/T/T	F/T/F/F/F/F	T/T/T/F/F/F	F/F/F/F/F/F	F/F/F/F/F/F
⑩	T/T/T/F/F/F	T/F/F/T/F/T	T/F/F/F/F/F	F/T/T/F/F/F	T/T/F/T/F/T	F/T/F/F/F/F	T/T/T/F/F/F	F/F/F/T/F/T	F/F/F/T/F/F
⑪	T/T/T/F/F/F/F	T/F/F/T/F/F/T	T/F/F/F/F/F/F	F/T/T/F/F/F/F	T/T/F/T/T/T/T	F/T/F/F/F/F/F	T/T/T/F/F/F/F	F/F/F/F/F/T	F/F/F/F/F/F
⑫	T/T/F/T/T/F/F/F	F/F/F/F/F/F/T	F/F/F/F/F/F/F	F/F/F/T/T/F/F	F/F/F/T/F/T/T	F/F/F/T/F/F/F	T/F/F/T/T/T/F	F/F/F/F/F/F	F/F/F/F/F/F

结合真值表和 5 种覆盖标准的定义，不准找出能够满足各种覆盖标准的测试用例，详细过程兹不赞述。

【答案】

判定覆盖 ③

条件覆盖 ②

判定/条件覆盖 ⑤

条件组合覆盖 ⑩

路径覆盖 ⑤

试题 3 （1998 年试题 3）

阅读以下说明和流程图，如图 6-14 所示回答问题 1 至问题 2，将解答写在答卷的对应栏内。

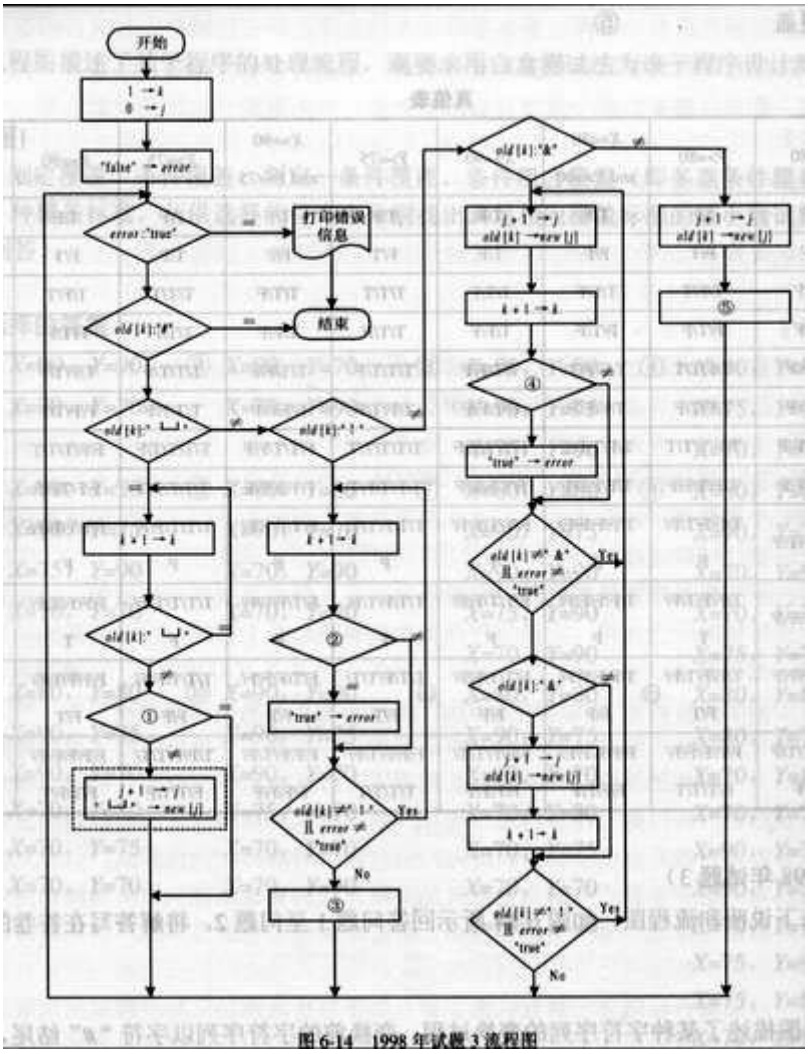


图 6-14 1998 年试题 3 流程图

［说明］

本流程图描述了某种字符序列的变换过程。变换前的字符序列以字符"#"结尾，它存放在数组 old 中，变换后的字符序列存放数组 new 中，它不含结束标志"#"。流程图按下列规则进行变换：

1、删除字符序列中的注解。注解用一对"！"字符分隔，注解中可出现除"！"和"#"外的任何字符。

2、字符常数完整地保留。字符常数用一对"&"字符分隔，字符常数中可出现除"#"外的任何字符。值得注意的是，当子符"&" 需要出现在字符常数中时，通常用两个连续的"&"表示，在保留字符常数时，这两个连续的"&"都被保留。此外，作为字符常数分隔符的一对"&"也被保留。

3、除字符常数和注解外，出现在其他地方的连续多个空格压缩成一个空格，但字符序列中先导的空格则全部删除。

4、注解和字符常数之外的非空格字符均保持不变。

本流程图对原字符序列从左到右扫描，根据遇到的当前字符来搜连续空格、注解或字符常数，然后按上述变换规则变换。若遇到当前字符是"！"，则寻找下一个"！"字符（若找不到则进行出错处理），这两个"！"间的字符全部看作注解。若当前字符是"&"，则寻找与之配对的下一个"&"字符（若找不到则作出错处理），其间的字符全部看作字符常数。

例如：

原字符序例：␣ ␣a␣ ␣ ␣b&cd&& ␣ ␣!e&f ␣ ␣g!h& ␣ ␣i ␣ ␣j#

变换后的字符序列：a␣b&cd&& ␣ ␣!e&f ␣g␣j

本流程图假定在数组 old 中一定存在"#"字符。

［问题 1］

填充流程图中的①～⑤，把解答写在答卷的相应们置上。

［问题 2］

当原字符序列中注解的前后均是连续空格时，本流程图将注解前后的连续空格分别压缩成一个空格，删除注解后，将导致变换后的新字符序列出现两个连续的空格。如：

␣ ␣g!h&␣ ␣i!␣ ␣j

本流程图将变换成 g␣ ␣j

为使变换后的新序列中除字符常数外没有连续的空格，图中的虚线框需作何改动（只需画出修改后的流程图）。

【解析】

程序流程图试题的解题关键在于把流程图看作流程图说明的实现，结合流程图说明理解流程图，确定流程图中各部分与流程说明的对应关系，然后再进行解答就非常容易了。

仔细阅读流程图，发现流程图大致分成 4 个纵列，其中第 1 纵列处理空格问题，第 2 纵列处理注释问题，第 3 纵列处理有关"&" 符号的问题，第 4 纵列处理正常字符。这样，我们就可以把整个流程图分成 4 个部分，在解答某一部分中的空白是时可以完全不考虑其他 3 个部分，大大降低了解答难度。

首先，关于空格处理的流程图说明包括两点，一是除字符常数和注解外，出现在其他地方的多个空格压缩成一个空格，二是字符序列中先导的空格全部删除。从流程图的"开始"处开始阅读，当读到 old [k]：［空格］处时，我们知道从此以下就是关于空格问题的处理，在 old [k]：为空格的情况下，k 值递加，续继比较 old [k]：是否为空格。当发现当前字符并非空格时，下面是填空①，我们暂且不管这个判断情况下可以跳过这步操作呢？即以上空格为多个空格压缩成一个空格的操作。究竟在什么情况下可以跳过这步操作呢？即以上空格为字符序列的先导空格时，可以跳过空格压缩的操作，所以该空白应该是对前位置的判断。从流程图中我们可以非常容易地确定 k 为数组 old 当前位置的指针，j 为数组 new 当前位置的指针。很明显，通过对 k 的判断不能确定处理过的空格是否为先导空格的，而当处理过的空格为先导空格时，数组 new 的当前位置应该动为 0，所以在填空①处填写"j:0"就是比较容易的事情了。

在这里可以先解答问题 2，因为问题 2 也是关于空格处理的，与流程图的基他部分没有关系。在读懂了流程图的第 1 纵列之后，应趁机这个问题 2。形成连续空格的原因是同于注解的阻隔，但我们绝对不能从这个原因入手来解决这个问题，那将使修改后的流程图规模膨胀，使本来简单的问题复杂化。问题的关键是数组 new 中不能出现两个连续的空格。虽然我们不能从原因入手解决问题，但却可以控制结果，即当需要往数组 new 中写入空格时，先判断数组 new 中当前字符是否为格。如果当前字符为空格，就不再写入空格，而是继续处理以后的字符。有了上述的分析，问题的解答就是非常容易的事情了。这只是一个思想方法的问题，问题的本身并没有难度，并键就是在考虑解决问题的方案时要从不同的角度考虑，找出最简单可行的办法。

填空②是在处理注解问题的纵列内，从流程图常识我们可以判断填空②应该填写一个判断语句，而且后续语句在该判断为真值时完成为出错标志赋真值的操作。通过阅读流程图说明可以知道，在处理注解时，唯一导致出错的情况就是注解中出现标志字符序列结束的符号"#"，所以该处的正确解答就是非常明显的，即时判断数组 old 的当前字符是否为字符序列结束标志"#"。填空④的解答思路与此相同。

填空③处的填写内容有一定的难度，但它的难度就在于此前的判断比较麻烦，如不仔细阅读很难搞清楚这些判断的含义，但只有弄明白了上述判断的含义，就不难判定该空白是在一处注解处理完毕之后，继续做处理的准备工作。跳过该空白继续阅读流程图，发现再次进入字符处理之后仍旧要判断组 old 的当前字符是否为空格、″！″或″&″。如果在空白处不能实现指针的向后偏移，则容易把一处注解的结束标志当作另一处注解的开始标志，造成程序的判断和处理错误。填空⑤的解答思路与此相同，这里就不再多说了。

通过对本流程图的解答，可以大致地总结以下有关解答此处类题目的思想方法，即紧密结合流程图说明，理解流程图对说明的实现，则很容易找出流程图说明中提出的而在流程图中没有实现的要求，然后结合对流程图的理解，在正确的位置上填写实现这些要求的语句。

【答案】

[问题 1]

①j:n

②old[k]:″#″

③k+1→k

④old[k]:″k″

⑤ k+1→k

[问题 2]

如图 6-15 所示.

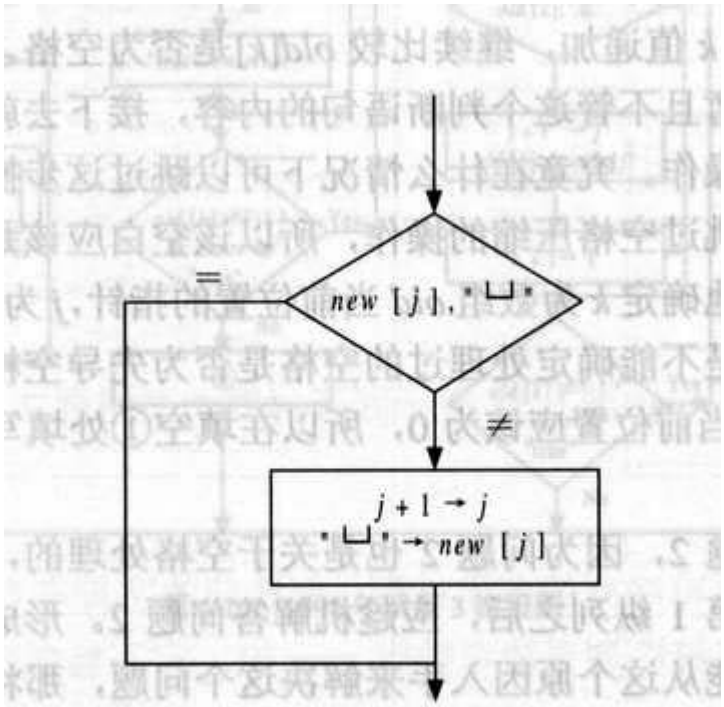


图 6-15 1998 年试题 3 问题 2 答案

6.4 1997 年度软件设计试题解析

1997 年度共 3 道软件设计题。试题 1 是一个数组处理的问题，试题说明很详细，没有涉及算法问题，这种情况在历年的试题中很常见。值得注意的是问题 2 和问题 3，这里没有涉及流程图的修改，而一个具体实例的运算，解答该类题目的关键是对处理流程的掌握，也即对流程图的全面深入掌握，而不是对流程图的修改和评价。

试题 2 考得比较细，主要涉及了 3 个方面的内容；（1）文件在系统中的作用；（2）操作的内容；（3）提高效率的途径。以上 3 个方面的内容在以前的试题中都出现过，要注意题中问题 2 的设计。问题 2 基本上涵盖了上述 3 个方面，是一个比较巧妙的问题，对考生的综合能力的要求比较高。

试题 3 考查了分层的数据流程图， 这方面的内容和软件测试一样，出现的频率较小。其实在解答这类试题时，只要能够贯彻逐步细化和数据平衡的原则，应该是相当容易的。

试题 1 （1997 年试题 1）

阅读以下说明和流程图，如图 6-16 所示，回答问题 1 至问题 3，将解答写在答卷的对应栏内。

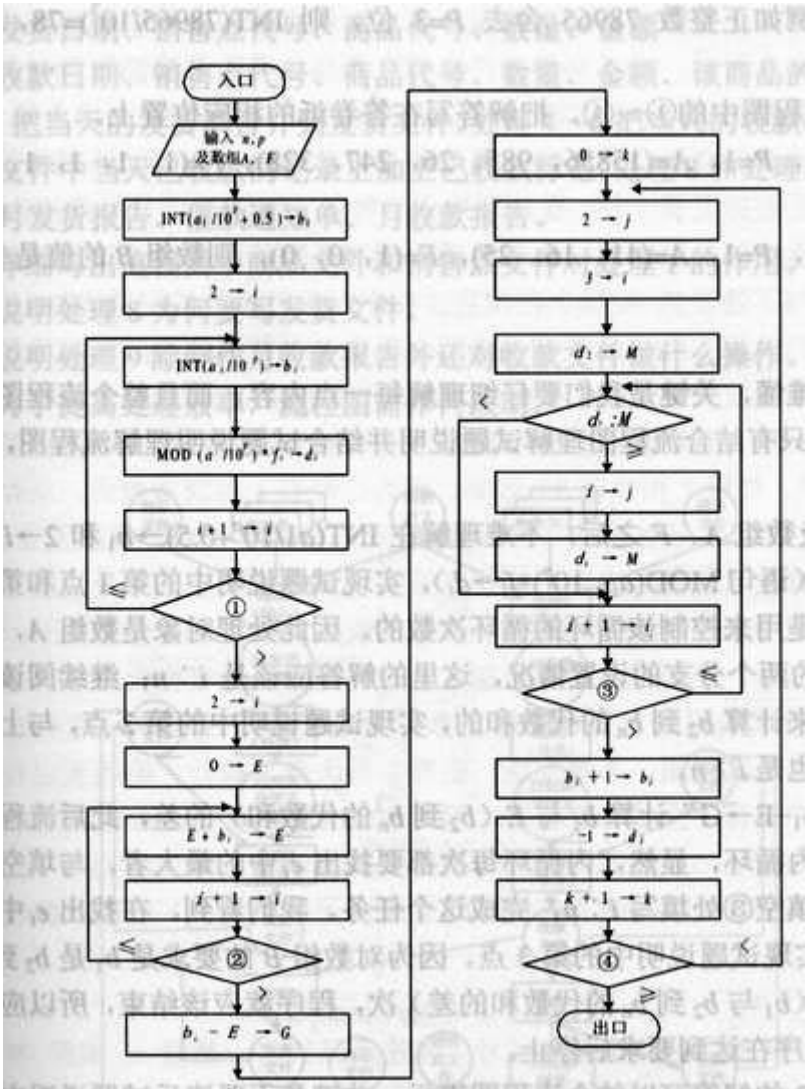


图 6-16 1997 年试题 1 流程图

［说明］

本流程图用来实现一组正整数的加权舍位平衡。已知正整数数组

A(a1a2……an)，满足条件 $a_1 = \sum_{i=2}^{n} a_i \quad (n \geq 3)$ 。现将数组 A 中每个数舍去 P 位，得到另一正整数数组 B（b1，b2……bn）。

它满足如下条件：

1.b1 是 a1 是舍 P 位后四舍入五所得，即 b1=INT（a1/10p+0.5）。

2. $b_1 = \sum_{i=2}^{n} b_i$

3.bi=INT(a1/10^P)+ei(i=2,3……，n)其中 ei 取值为 0 或 1，当 e1=时，称 ei 第 i 项数据的进位。

4.ei（i=2，3 ……，n）之值根据余数 MOD(ai,10^P)乘上权 fI（f≥0）后的数值大小来决定（其算法见图 15-9 所示的流程图）。权 fi 存放在数组 f 中，其中 INT 是取函数，MOD 是余数函数，例如正整数 7895 舍去 P=3 位，则 INT（78935/10^3）=78，MOD（7895，10^3）=965。

［问题 1］填充流程图中的①～④，把解答写在答卷纸的相应位置上。

［问题 2］若 N=5，P=1 A=（15856，985，26，247，328），F=（1，1，1，1 ，1）则数组 B 的值是多少?

［问题 3］若 N=3， P=1，A=（41，16，25）F=（1，2、0，0）则数组 B 的值是多少?

【解析】

试题说明并不难懂，关键是我们要仔细理解每一点内容，而且整个流程图实现了试题说明的 1～4 项内容，只有结合流程图理解试验题说明并结合试题说明理解流程图，才能真正理解试题说明和流程图。

在输入 n、p 及数组 A、F 后，不难理解在 INT（a1/10^P+0.5）→b1 和 2→i 之后，循环计算 bi 的基本值和 ei（语句 MOD（ai,10^P）*fi-di），实现试题说明中的第 1 点和第 4 点要求的操作。显然，填空合①是用来控制该循环次数的。因此处理对象是数组 A，所以其元素个数为 n。结合判决的两个分支的设置情况，这里的解答应该是 i:n；继续阅读流程图，填空②之前的循环是用来计算 b2bn 的代数和的，实现试验题说明中的第 2 点，与上面的解答过程类似， 这里的判断也是 i:n。

显然，语句"b1→E→G"计算 b1 与 E（b2 到 bn 的代数和）的差，此后流程图进入一个二重循环。首先研究内循环，显然，内循环每次都要找出 ei 中的最大者，与填空①、②的解答相似我们在填空③处写 i:n，完成这个任务。我们看到，在找出 ei 中的最大者后，相应的 bj 即加 1，实现试题说明中的第 3 点。因为对数组 B 的要求是 b1 是 b2 到 ba 的代数和，当加 1 操作进行 G(b1 与 b2 到 bn 的代数和的差)次，程序就应该结束，所以应在空④处填写 k:G，以控制程序在达到要求后停止。

问题 2 和问题 3 的解答可结合流程图进行，关键是不要违反试题说明中的各种要求，解答的难度应该不是很大。

【答案】

［问题 1］①i:n

［问题 2］159，98，3，25，33

［问题 3］4,1,3

试题 2 （1997 年试题 2）

阅读以下说明和流程图 6-17,回答问题 1 至问题 4,将解答写在答卷的对应栏内。

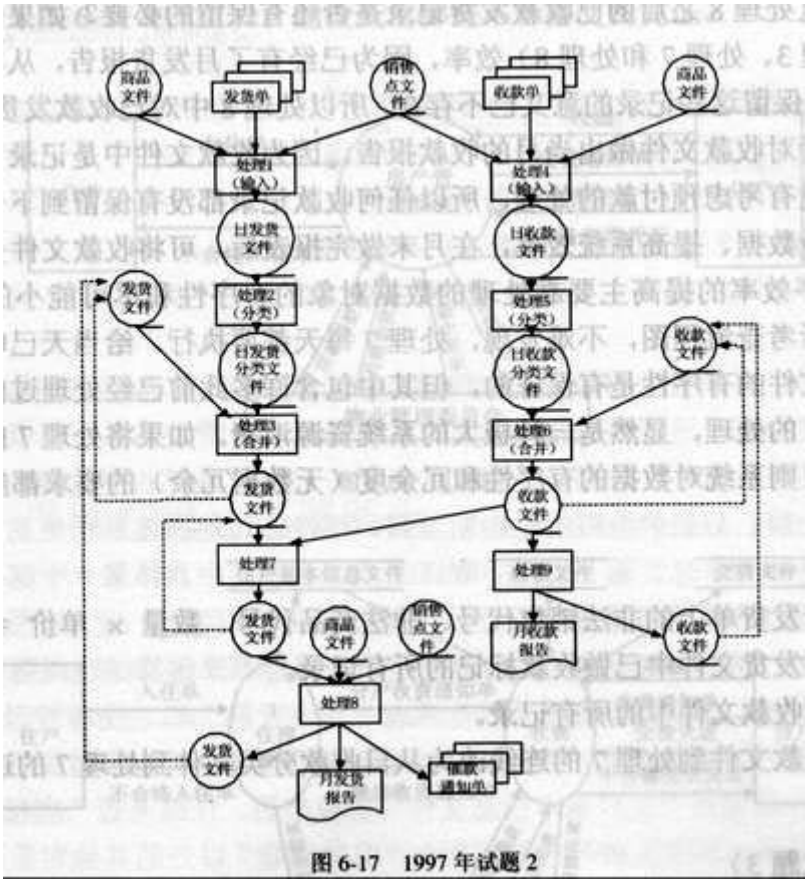


图 6-17 1997 年试题 2

［说明］

某公司将其生产的商品通过若干销售点进行销售。销售点在收到商品后的规定时间内把货款江给公司。

流程图描述了该公司发货、收款、催款的处理过程。其中部分文件和单据的格式如下。

商品文件：商品代号、商品名称、单价

销售点文件：销售点代号、销售点名称、地址

发货单：发货日期、销售代号、商品代号、数量、金额

收款单：收款日期、销售代号、商品代号、数量、金额、该商品的发货日期

处理 1～3 把当天的发货单合并到发货文件。处理 4～6 把当天的收款单合并到收款文件。处理 7 在发货文件中当天已收款的记录确良加上已收款标记。处理 8 和处得 9 在月末执行一次，主要用于输出月发货报告、催款通知单、月收款报告。

［问题 1］详细写出流程图中商品文件和销售点文件对处理 1 的作用。

［问题 2］说明处理 8 为何要写发货文件。

［问题 3］说明处理 9 除制作月收款报告外还对收款文件做什么操作。

［问题 4］为了提高处理效率，流程图需作何改动。

【解析】

本题的流程图描述了某公司发货、收款和催款的处理过程。通过阅读题目所给出的流程图可以了解到下面几点内容：流程图中描述的一方面是对发货单的日常处理，另外一方面是对收款单的日常处理，两者的交叉点是每天在发货文件中当天已收款的记录上加上已收款标志，还有每月对发货文件和收款文件的报表打印和催款通知单的打印。

一般来说，在对数据输入和处理的过程中都要进行数据的合法性和一致性的检查。本题的流程图中处理 1 和处理 4 都是数据输入处理，这一过程必然要进行数据合法性的检验。在这里合法性的检查主要有在将发货单合并成日发货文件的时候，要检查发货单中数据的正确性，即发货单中商品代号、销售代号、数量以及金额的正确性。那么这里是如何进行合法性检查的？考查商品文件、销售点文件及发货单的数据项，不难推断，对销售点代码的检查可以通过在销售点代码录入有误。通过商品文件来检查商品代码的合法性原理与相应的记录，说明发货单中数量和金额两个数据项的检查也可以通过商品文件来实现，如果金额/数量≠单价，则说明数量或金额的录入有误。这里的前提是商品文件和销售文件是准确无误的。

处理 8 的内容是根据发货文件、商品文件和销售点文件来生成月发货报告、催款单文件。究竟为什么要写发货文件，这要通过此前此后发货文件的使用来解答。很明显，发货文件在参与处理 8 之前参与了处理 7，处理 7 在发货文件中当天已收款的记录加上了已收款标记。这样，在处理 8 中的发货文件记录可以分成两个集合，一是已收款的发货记录，一是未收款的发货记录。未收款的发货记录是生成催款单通知的根据，已收款的发货记录，一是未收款的发货记录。未收款的发货记录是生成催款单通知的根据，已收款的发货记录是生成月发货报告的根据。经过处理 8 之后的已收款发货记录是否还有保留的必要？如果保留，必须影响此后的处理（处理 3、处理 7 和处理 8）效率，因为已经有了月发货报告，从存留档案资料的角度看，在系统中保留这些记录的意义已不存在，所以处理 8 中对已收款发货记录进行删除。

处理 9 则是针对收款文件做出当月的收款报告。因为收款文件中是记录当月的收款单，而且在流程图中没有考虑预付款的处理，所以任何收款记录都没有保留到下一个月的必要。据此，为消除冗余数据、提高系统效率，在月末做完报表后，可将收款文件全部清空。

流程图/程序效率的提高主要靠处理的数据对象的有序性和尽可能小的数据冗余来实现。结合以上分析查流程图，不难发现，处理 7 每天都要执行。给当天已收款的记录加已收款标志，收款文件的有序性是有保证的，但其中包含许多此前已经处理的记录，每天都要参与一次处理 7 的处理，显然是一种极大的系统源浪费。如果交处理 7 的数据来源改为日收款分类文件，则系统对数据有序性和冗余度（无数据冗余）的要求都能够得到满足。

【答案】

［问题 1］检查发货单上的非法销售代号、非法商品代号、数量×单价≠金额等错误。

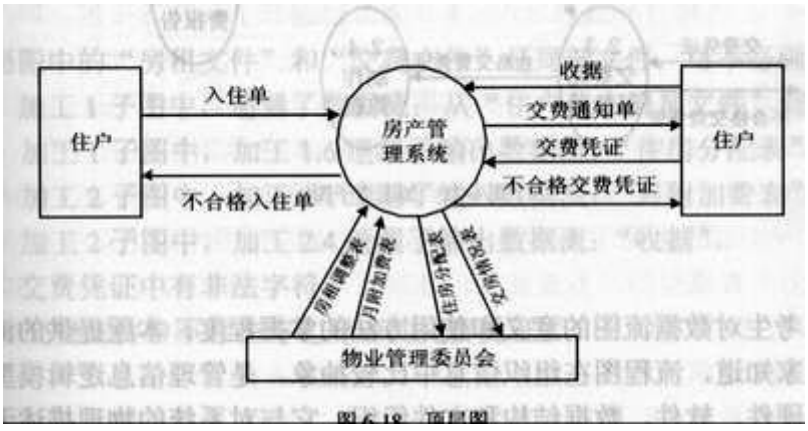
［问题 2］删除发货文件中已做收款标记的所有记录。

［问题 3］删除收款文件中的所有记录。

［问题 4］从收款文件到处理 7 的连线改为从日收款分类文件处理 7 的连线。

试题 3 （1997 年试题 3）

阅读以下说明和流程图（如图 6-18 至图 6-21 所示），回答问题 1 至问题 3，将解答写在答卷的对应栏内。



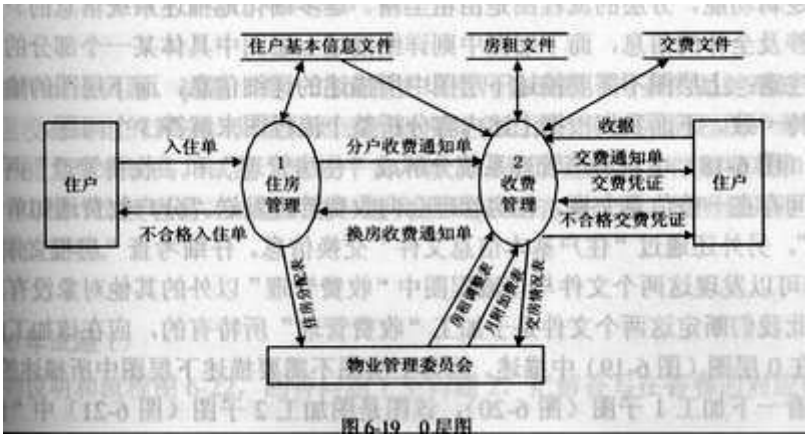


图 6-19 0 层图

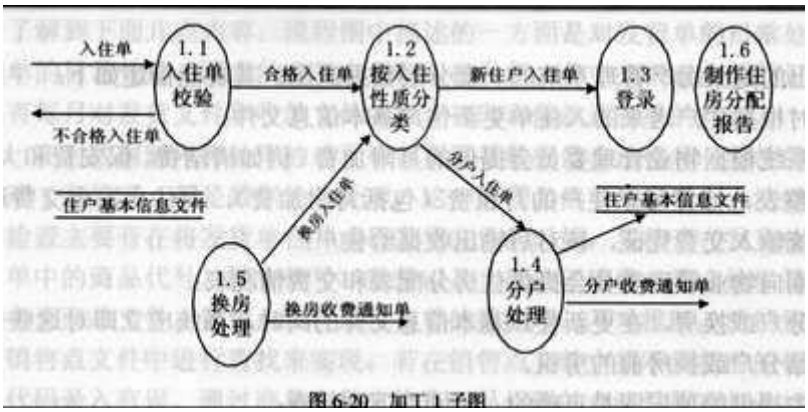


图 6-20 加工 1 子图

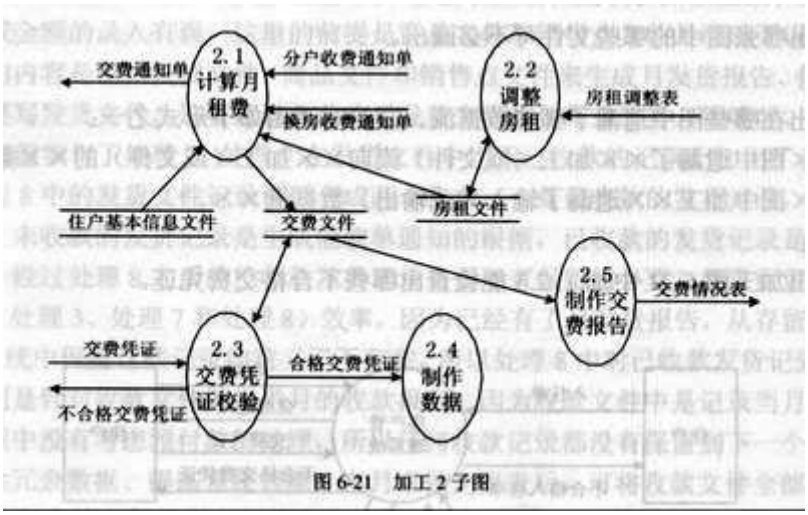


图 6-21 加工 2 子图

［说明］

下面给出的是某房产管理系统的一套分层数据流图。其功能描述如下：

系统随时根据住户送来的入住单新住户基本信息文件。

每月初系统根据物业管理委员会提供的月附加费（例如清洁费、保安费和大楼管理费等）表和房租调整表，计算每家住户的月租费（包括月附加费），向住户发出交费通知单，住户交费时，系统输入交费凭证，核对后输出收据给住户：

系统定期向物业管理委员会提供住房分配表和交费情况表；

住户因分户或换房，在更新住户基本信息文件的同时，系统应立即对这些住户做月租费计算，以了结分户或换房前的房租。

假定题中提供的顶层图是正确的，请回答下列问题：

［问题 1］

指出哪张图中的哪些文件可不必画出。

［问题 2］

指出在哪些图中遗漏了哪些数据流。回答时请用如下形式之一：

XX 图中遗漏了 XX 加工（或文件）流向 XX 加工（或文件）的 XX 数据流；

XX 加工 XX 遗漏了输入（或输出）数据流 XX；

［问题 3 ］

指出加工图 6 — 2 1 中加工 2． 3 能检查出中些不合格交费凭证。

【解析】

本题是考查考生对数据流图的意义和使用方法的掌握程度。本题提供的流程图是一种分层的流程图。大家知道，流程图在组织信息中比较抽象，是管理信息逻辑械型的主要形式，这人模型不涉及硬件、软件、数据结构和文件组织，它与对系统的物理描述无关，只是表不一种系统的逻辑功能。分层的流程图是由粗至精、逐步细化地描述系统信息的。上层图中描述的是粗略涉及全体的信息，而下层图中测详细描述上层图中具体某一个部分的内容。另外我们一定要注意：上层图不需要描述下层图中所描述的详细信息，而下层图的输入与输出应与上层图保持一致。下面我们根据上述内容分析整个流程图来解答 3 个问题。

顶层图（图 6-18）中将房产管理系统分解成"住房管理"和""收费管理两个子加工，两个部分之间存在一些信息交换。住房管理会向收费管理发送"分户收费通知单"和"换房收费通知单"，另外还通过"住户基本信息文件"交换信息。仔细考查"房租文件"和交费文件"，我们可以发现这两个文件与该流程图中"收费管理"以外的其他对象没有其他任何信息交流，由此我们断定两个文件是子加工"收费管理"所特有的，应在该加工的子图中描述，而不应在 0 层图（图 6-19）中描述。因为上层图不需要描述下层图中所描述的详细信息。

我们再看一下加工 1 子图（图 6-20），该图是图加工 2 子图（图 6-21）中"住房管理"的具体细化，它分为 6 个子加工，即入住单校验、按入住性质分类、登录、分户处理、换房处理和作住房分配报告。从流程图中可看出，"入住单校验"没有借助任何外部数据，不借助外部数据的校验是无法想像的，除非入住单数据项上有某些固定的对应关系，但似乎是不可能的。首先考虑入住单中会有什么错误，不合格的入住单无非就是对住房的重复分配。"住房管理"加工中所有的外部数据源，应根据"住户基本信息文件"来校验入住单，所以加工 1.1 缺少由"住户基本信息文件"流向它的数据流。根据 0 层图（图 6-19）中加工的输入输出流来检查加工 1 子图（图 6-20），与 0 层（6-19）中的输入流和输出流相比较，加工 1 子图（图 6-20）中缺少了"住房分配表，"这一输出流，分析加工 1.1 至加工 1.6，可知制作住房分配报告由加工 1.6 负责，因此断定加工 1.6 缺少了输出数据流"住房分配表"。

接下来分析加工 2 子图。该图是顶层图"收费管理"的具体细化，该加工分为 5 个子加工，即月租费、调整房租、缴费凭证校验、制作收据和作交费报告，并且附加了两个文件，即文件和交费文件。同样我们也看一下 0 层图（图 6-19）的输入输出数据流：输入流"分户收费通知单"、"换房收费通知单"、"交费凭证"、"房租调整表"都存在，但少了一个"月附加费表"，这个表是在计算月租费的时候使用，所以加工 2.1 缺少了输入数据流"月附加费表"。输出流"交费情况表"、"不合格交费凭证"、"交费通知单"都存在，但缺少了"收据"这一项，而"收据"应由加工 2.4 产生，因此加工 2.4 缺少了输出数据流"收据"。

在子加工 2.3 中可以检查出的不合格交费凭证是：①是凭证的语法错误，也就是输入信息不合规范；②是输入的数据虽然合乎规范，但是内容不对（参照原解答修改此处）。

【答案】

[问题 1] 0 层图中的"房租文件"和"交费文件"是局部文件，可不必画出。

[问题 2]①加工 1 子图中，遗漏了数据流：从"住户基本信息文件"到加工.1.1。

②加工 1 子图中，加工 1.6 遗漏子输出数据流："住房分配表"。

③加工 2 子图中，加工 2.1 遗漏了输入数据流："月附加费表"。

④加工 2 子图中，加工 2.4 遗漏了输出数据流："收据"。

[问题 3] ①交费凭证中有非法字符。

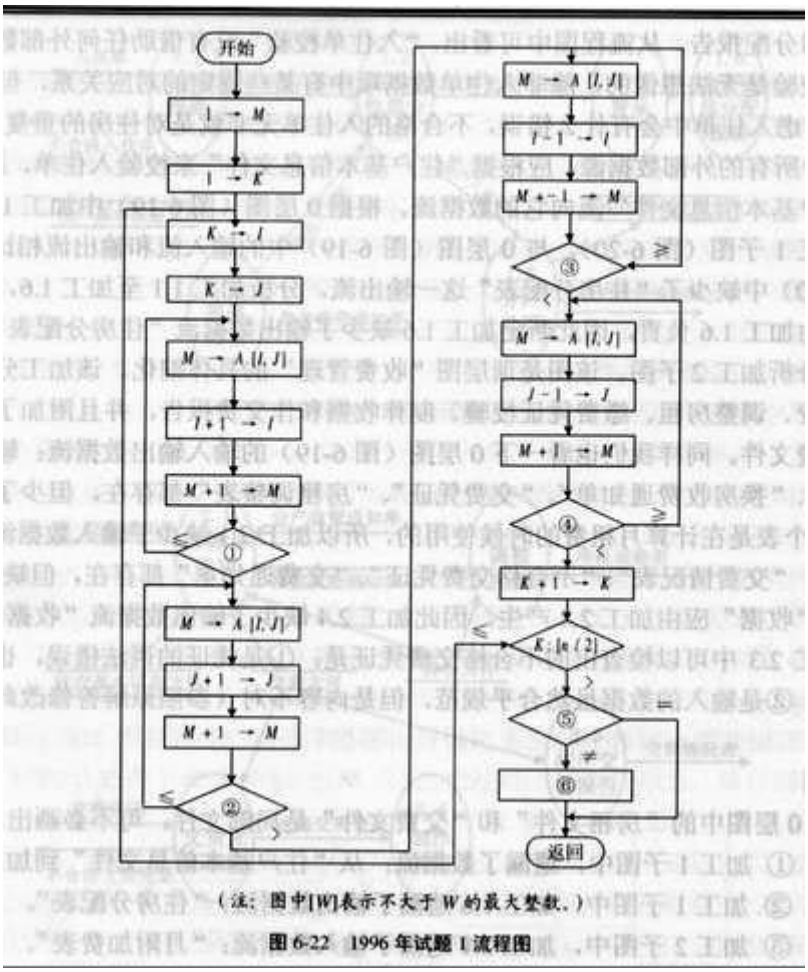
②交费文件中不存在与之对应的交费凭证。

6.5 1996 年度软件设计试题解析

共 3 道软件设计试题。试题 1 是一道流程图填空修改题，考查的是蛇形矩陈的生成，很明显，这里没有固定的算法，只能依照试题说明提供的两个示例从流程图中寻找算法的实现，对考生的能力要求较高。试题 2 比较平实，涉及了错误查找、文件数据项和提高处理速度的途径 3 个方面，比较容易。试题 3 是一道关于 SQL 语言的度题，很明显是注重考查考生对流程图总体的把握。

试题 1 （1996 年试题 1）

阅读下列说明和流程图 6-22，回答问题 1 至问题 2，把解答写在答卷的对应栏内。



（注：图中[W]表示不大于 W 的最大整数。）

图 6-22 1996 年试题 1 流程图

［说明］

本流程图将数字 1，2……，N（N≥2）接逆时针方向依次写在 N×N 矩阵中。下图给出了 N=4 和 N=5 的情况：

1	12	11	10
2	13	16	9
3	14	15	8
4	5	6	7

（N=4 时）

1	16	15	14	13
2	17	24	23	12
3	18	25	22	11
4	19	20	21	10
5	6	7	8	9

（N=5 时）

［问题 1］
填充流程图中的①～⑥，使之成为完整的流程图。

［问题 2］

若将数字 1，2……，N2 按顺时针方向依次写在 N×N 矩阵中，则只需将上述流程图中的⑦改成⑧即可。

【解析】

粗读流程图可知，程序开始先初始化 M 和 K 为 1，然后就进入一个大循环。在这个大循环中，包括 4 个循环①、②、③、④。在循环①中 J 值不变，I 值每循环一次加 1，即列号不变，行号递增，是从上往下将值写入矩阵中；在循环③中 J 值不变，J 值每循环一次减 1，即列号不变，行号递减，是从下往上将值写入矩阵中；在循环④中 J 值不变，I 值每循环一次减 1，即行号不变，列号递减，是从右往左写入矩阵中；4 个小循环结束以后，K 值加 1。

由于本题的功能是将自然数按逆时针方向依次写在 N×N 矩阵的一圈，而外层循环的功能则是控制里面循环的次数，即总共写几次，也即 N×N 矩阵总共有几圈，所以 K 的功能的功能则是控制我层循环的次数，同时也控制写入第几圈及开始写入的位置。

而 M 的功能则相对简单，M 在每个小循环中都是递增，且其值被赋给矩阵的元素，可知 M 为填入矩阵的值，它对循环没有什么影响。

在任一圈上依次按 4 条边写入矩阵时，每一边的起点都在对角线上，终点都不在对角线上，每一边要写入的元素的个数都相同，当前的圈数为 K，则每一边所应写入矩阵的元素的个数均为 N-24K+1，即写入每一边的小循环次数应为 N-2K+1。

再仔细阅续每个小循环的流程图。循环①的功能是从上往下写数，那么其结束的标志应该是已经把这一列写完了，该边行的起始位置为 K，所以结束条件就为 I：N-K，这样总的循环次数为 N-2K+1。

循环②的功能是从左右写数，那么其结束的标志应该是已经把这一行写完了，该边列的起始位置为 K，所结束条件就为 J：N-K，这样总的循环次数为 N-2K+1。

循环③的功能是从下往上写数，那么其结束的标志应该是已经把一行写完了，该边行的起始位置为 N-K+1，所以结束条件就为 I：K+1，这样总的循环次数为 N-2K+1。

循环④的功能是从右往左写数，那么其结束的标志应该是已经把这一行写完了，该边行的起始位置为 N-K+1，所以结束条件就为 J：K+1，这样总的循环次数为 N-2K+1。

外循环每循环一次，K 值加 1，若 K 大于 N/2，此时已达到矩阵中心，则外层循环结束；否则尚未达到矩阵中心，则从 A［K，K］开始重复上述操作。若已达到矩阵中心，又有两种情况：若 N 为偶数，则所有的元素均填写完毕，程序结束；若 N 为奇数，则矩阵中心元素所以⑤的答案为 MOD（J，2）：0 或其他任何能判断出 N 的奇偶性的表达式。

观察矩阵可知，只要将按逆时针方向写入得的到的矩阵置，即将矩阵的行列互换，就得到将字 1，2……， N2 按顺量针方向写入得到的矩阵。在流程图中，I 中代表行，J 代表列，将 J 与 I 互换即可达到行到互换的目的，所以只要将 A［I，J］改成 A［I，J］即可得到按顺时针方向写成的 N×N 矩阵。

【答案】

［问题 1］

①I:N-K 或 I:N-J 或 M:4(K-1)(N-K+1)+N-2K+1

②J:N-K 或 J:I-1 或 M:4(K-1)(N-K+1)+2(N-2K+1)

③I:K+1 或 I:N+2-J 或 4(K-1)(N-K+1)+3(N-2K+1):M

④J:K+1 或 J:I+1 或 4K(N-K):M

⑤MOD(N,2):0 或 [N/2]*2:N 或 [N/2]:[N+1/2] 或 N/2: [N/2] 或 [N/2]:[N/2] 或 N:2(K-1) 或 I+N=N 或 M-1:N^2

⑥→A[K,K]

［问题 2］

⑦A[I,J] 或 M→A[I,J]

⑧A[J,I] 或 M→A[J,I]

试题 2（1996 年试题 2）

阅读下列说明和流程图 6-23，回答问题 1 至问题 3，把解答写在答卷的对应栏内。

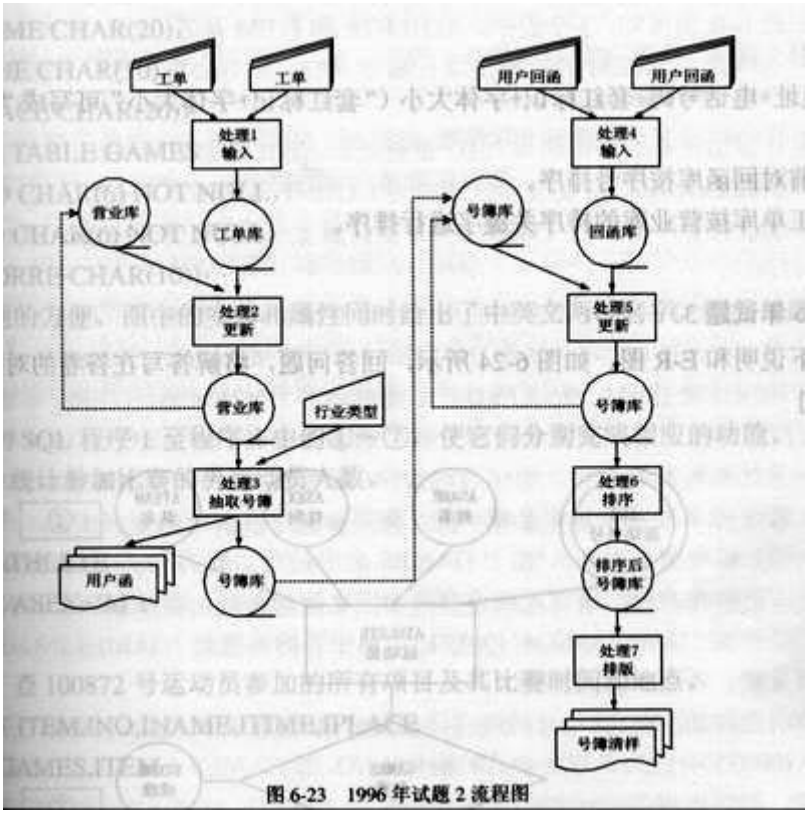


图 6-23 1996 年试题 2 流程图

［说明］

本流程图描述了某行业分类电话号码薄（简称号薄）出版系统的处理流程。全市所有电话的基本信息均存入在营业库中。系统输入工单，工单中包括电话的新装、拆除、移机和更改（更改户名、地址和电话号码等）信息。为确保输入工单的正确性，每张工单均由两个录入员分别录入，由处理 1 进行输入和校对，然后理更新营业库。系统根据特出版号薄的行业类型从营业库中选取该类用户的电话信息，存入在号薄库中。 同时向每个电话用户发出用户函，用户函上记录着半刊登在号薄库中的位置。用户收到用户函后，进行校对，并将修改内容和印刷要求（字体大小和是否套红）填写在用户回函中，系统按收到用户回函的先后顺序依次输入用户回函，然后更新号薄库，最后通过排版输出经用户校对并符合其印刷要求的号薄清样。

系统中部分单据和文件的格式如下：

工单=工单类型=户名+地址+电话号码

营业库记录=户名+地址+电话号码+分类信息

用户函=序号+户名+地址+电话号码

用户回函=序号+户名+地址+电话号码+套红标记+字体大小

［问题 1］

流程图中哪些处理能发现工单的哪些错误，并举例说明。

［问题 2］

指出号薄库文件的记录至少应包括哪些据项。

［问题 3］

为提高和理速度，流程图需作何改进。

【解析】

在哪些处理中能发现哪些错误,也就意味着要在哪些处理中检查哪些错误。一般来说，在数据进入系统后需要马上进行数据合法性检查，及时排除错误数据，保证以后的处理有正确的数据来源。考查流程图，可以看出处理 1 和处理 4 完成数据录入员录入工单，并对录入的数据进行核对的。此时可以发现两人录入的不同之和和录入的非法数据,核对后的数据放入工单库中。处理 4 由两个录入员入用户回函信息，核对后放入回函库，这一步也可以发现两个录入员输入不同信息和输入非法数据。但是我们不要忽视处理 2 处理 2 根据营业库中已有的用户信息对照工单营业库进行更新。这一步可以发现工单库与营业库数据不一致之处，例如工单库中有一人顾客要安装电话，而营业库中却发现该电话码已存在，即一个电话号码的重复分配。

为考查号簿库包含的数据项，我们需要考查号簿库的数据来源的的去向。号簿的数据来源是营业库，而营业库的数据项是由工单而来，但因为号簿库与工单已经没有直接联系，所以能够断定营业库中的数据项是可以包含在号簿库数据项中的。是否需要全产包含，尚不能断定。用户函是由号簿库而来的，我们推敲用户函在以后处理中的使用情况，可以推知在用户函中每个用户可以选自己的信息印刷格式。如字体大小和套红标记，所以号簿文件中还应含有印刷要求，如字体大小和套红标记。现在再考虑号簿库中是否需要分类信息？显然是不需要的，因为号簿库在生成时已经是按分类存放了（参见处理 3），所以分类信息可以从号簿库中排除出去。

从软件上提高数据库处理速度最常用性的方法是保证数据的有序性，因此我们可以在处理 5 前对回函库按序号排序，这样可以提高处理 5 的更新速度，节省处理时间。此外，还可以在处理 2 前对我单库按营业库的排序关键字进行排序，这样可以加快营业库更新的速度，节省更新时间。

【答案】

［问题 1］

处理 1 能发现两个人录入不全同的工单和非法数据（如非法字符和数据越界等）。

处理 2 能发现与营业库不一致的工单（如新装电话重号、移机、拆机、更改和原电话号码不存在等）

处理 4 能发现两人录入不全同的用户回函和非法数据（如非法字符和电话号码长度不对等）。

［问题 2］

户名+地址+电话号码+套红标识+字体大小（"套红标记+字体大小+可写成"印刷要求"）。

［问题 3］

处理 5 前对回函库按序号排序。

处理 2 前对工单库按营业库的排序关键字进行排序。

试题 3 （1996 年试题 3）

阅读以下说明和 E-R 图，如图 6-24 所示，回答问题，将解答写答卷的对应栏内。

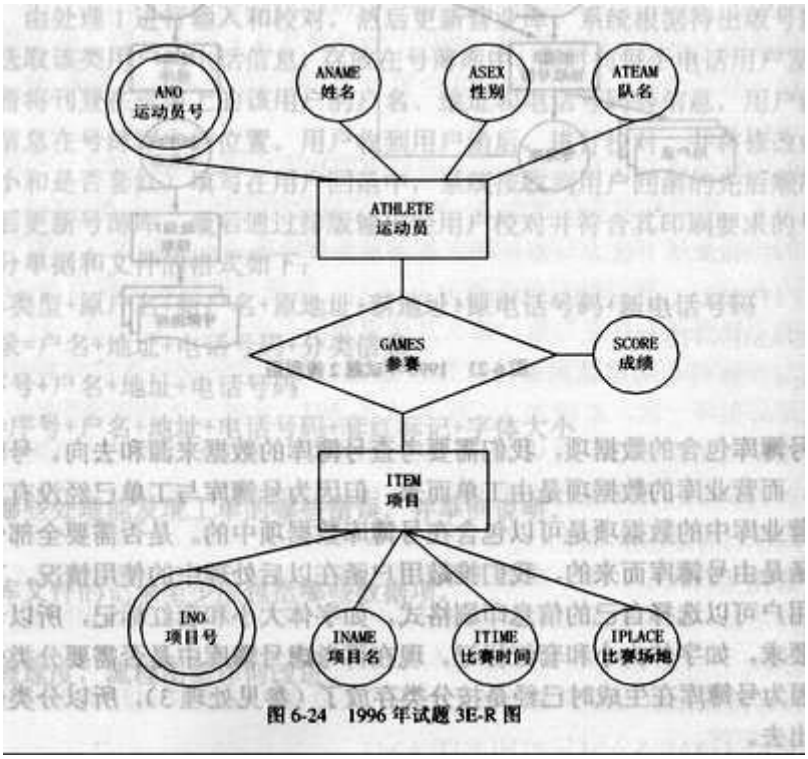


图 6-24 1996 年试题 3E-R 图

［说明］

设有下列关于运动会管理系统的 E-R 图。图中矩形表示实体，圆表示属性，双圆表示关键字属性，菱形表示实体之间的关系。假定已通过下列 SQL 语言建立了基本表：

```
CREATETABLEATHLETE
(ANOCHAR(6)NOT NULL,。
ANAMECHAR(20)
ASEXCHAR(1),
ATEAMCHAR(20);
CREATETABLEITEM
(INOCHAR(6)NOTNULL,
INAMECHAR(20),
IPLACECHAR(20);
CREATETABLEGAMES
(ANO CHAR(6)NOTNULL,
INOCHAR(6)NOTNULL,
SCORRECHAR（10）；
```

为了答题的方便，图中的实体和属性同时给出了中英文两种名字，回答问题时只需写出英文名即可。

［问题］

填充下列 SQL 程序 1 至程序 4 中①～⑦，使它们分别完成相应的功能。

程序 1：统计参加比赛的男运动员人数。

```
SELECT ①
FROM ATHLETE
WJEREASEX="M"
```

程序 2：查 100872 号运动员参加的所有项目及其比赛时间和地点。

```
SELECTIEM.INO,INAME,ITIME,IPLACE
WHERE ②
AND ③
```

程序 3:查参加 100035 项目所有运动员名单。

```
SELECTANO，ANAME，ATEAM
FROM ATHLETE
WHERE ④
SELECT ⑤
WHEREGAMES.ANO=ATHLETE.ANO
ANDINO=100035;
```

程序 4：建立运动员成绩视图

```
⑥ ATHLETE-SCRE
```

```
ASSELECTATHLETE.ANO,ANAME,ATEAM,INAME,SCORE
FROM ⑦
WHERE ATHLET.ANO=GAMES.ANO
ANDGAMES.INO=ITEM.INO;
```

【解析】

本题是关于系数据库标准语言--SQL（Structured Query Language）语言的题目，由题目中经给出的 E-R 图可知，3 个表中，ATHLETE 和 ITEM 是基本表，表 ATHLETE 的主键是运动员编号 ANO，表 ITEM 的主键是项目编号 INO，表 GAMES 是一个视图，以 ANO、INO 为外键。

程序 1 统计参加比赛的男运动员人数，也就是表 ATHLETE 中，AEX='M'的记录的个数，所以要用到库函数 COUNT（*）。这里注意的是 COUNT 与 COUNT（*）的区别，COUNT 的功能是对一列中的值计算个数，而 COUNT（*）。这里要注意的确良 COUNT 与 COUNT（*）区别，COUNT 的功能是对一列中的值计算个数，而 COUNT（*）才是计算数据库中记录的个数。所以填空①的答案为"COUNT（*）"。

程序的 2 统计 100872 号运动员参加的所有项目及比赛时间和地点，所以 SELECT 后面的内容是项目编号 ITEM.INO、项目名称 INAME 时间 ITIME 及地点 IPLACE。统计涉及到比赛表 GAMES 和项目表 ITEM，所以 FROM 后面的内容为 GAMES、ITEM。本题考的是连接查询，所谓连接查指的是涉及两个以上的表的查询。由于是统计 100872 号运动员参加的所有项目及比赛时间和地点，所以查询条件中必然有 GAMES.INO='100872'（程序中引用到字段时，若字段名在各个表中是唯的，则可以把字段名前的表名去掉，否则，应当加上表名作为前缀，以免引起混淆）。由于 GAMES 表中只有比赛的成绩，那些关于项目的数据必须从项目表 ITEM 中取得，所以还应该有两个表之间的关联，即 GAMES.INO=ITEM.INO。所以填空②和③可交换，不影响查询结果。

程序 3 要求查参加 100035 项目的所有运动员名单。分析查询表达式，必首先查 GAMES 表，找出参加 100035 项目的那些运动员的编号 ANO,即 GAMES.ANO=ATHLETE.ANO AND INO='100035',然后再根据查询到的运动员的信号 ANO 从 ATHLETE 表中取妯运动员的数据。所以填空④的答案为"EXISTS"或"ANOIN"，填空 ⑤的答案为"ANO"

程序 4 要求建立运动员成绩视图。建立视图的命令为 CREATEVIEW，所以填空⑥的答案一定是"CREATEVIEW"。建立的是运动员成绩视图，那么一定涉及运动员情况、运动员参加的项情况和该项目的成绩，所以要用到 ATHLETE、ITEM 和 GAMES 这 3 个表，因此 FROM 子句后为 ATHLETE、GAMES、ITEM，3 个表可以是任意次序，不影响结果。

【答案】

①COUNT（*）（若答 COUNT 或 COUNT*得 2 分）

②GAMES.INO=ITEM.INO

③GAMES.ANO='100872'（注：②、③可互换、无前缀得 1 分）

④EXISTS

⑤*或 ANO 或 INO 或 SCORE 或后 3 个列名的任意组合

⑥CREATEVIEW

⑦ATHLETE， ITEM， AMES（3 项可交换。）

注：④、⑤也可为
④ANOIN
⑤ANO

6.6 1995 年度软件设计试题解析

1995 年共有 5 道软件设计试题供选答。试题 1 总体说难度不大，问题 1 和问题 2 的形式是在广大考生的意料之中的，问题 3 涉及对流程图的评估，只要能够仔细理解系统处理流程，应该是比较容易的。流程图填空（试题 2、试题 3）分别涉及用枚举法录找符合要求的自然数与对紧缩存储的一元多项式相乘两个方面的问题，而且都涉及了对流程图进行修改的问题。应该说，这两道试题的考查重点是能力考查，如果不能迅速理解试题说明所提供的算法和流程图的结构，解答问题的难度将非常大。试题 4 采用状态转换矩阵来检验非空算术表达式的语法正确性，解题关键是要明确各个状态含义，这在客观上增大了答题的工作量，而且解题所需信息只能从试题说明中获得，这是一道纯碎的能力考查试题。试题 5 是一道关于软件测试的题目，很明显，这部分知识的概念是容易记忆的，但在知识的具体运有上却并非易事。本题是一道知识与能力考查并重的试题，能否成功解答本题关键是看知识掌握程度和解决的能力。

试题 1（1995 年试题 1）

阅读下列说明和流程图（如图 6-26 所示），回答问题 1 至问题 3，把解答写在答卷的对应栏内。

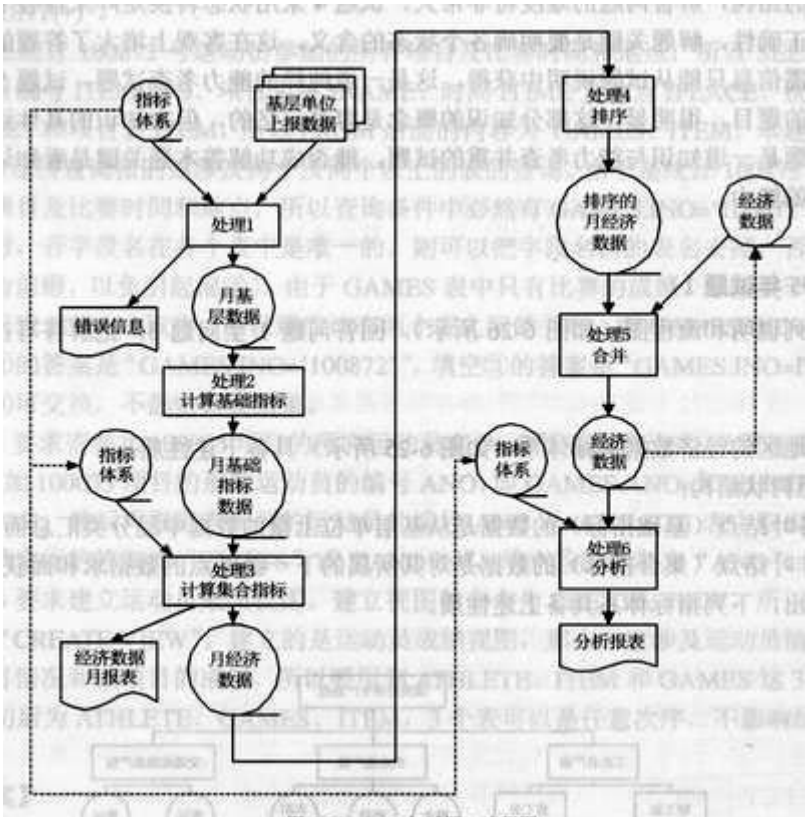


图 6-26 1995 年试题 1 流程图

[说明]

假定某地区的经济数据指标体系（如图 6-25 所示）具备下述性质：

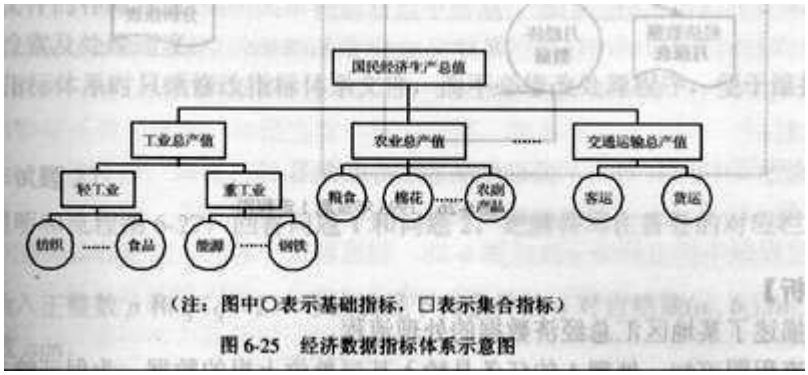


图 6-25 经济数据指标体系示意图

（1）呈树状结构：

（2）各叶结点（基础指标）的数据是从基层单位上报的数据中经分类汇总而获得；

（3）非叶结点（集合指标）的数据是对其所属的下一级结点的灵敏据求和而获得的。不难看出，下列指标体系具备上述性质。

例如：粮食、棉花……和农副产品产值（基础指标）可以从各县及各农场每月上报的数据中经分类汇总获得，农业总产值（集合指标）=粮食产值+棉花产值+……

农副产品产值。
本流程图用来计算月基础指标数据和月集合指标数据，产生经济数据月报表，并把月经济数据加载到经济数据文件中，产生分析报表。

假定有关的文件和单据的记录格式如下：

指标体系文件：指标代码、指标名称、计量单位

基层单位上报数据：单位名称、指标名称、产值

［问题 1］

简述处理 1 的处理内容。

［问题 2］

指出月基础指标数据文件的记录格式至少应包含哪些内容。

［问题 3］

简要叙述指标体系文件中的指标代码的主要作用。

【解析】

本题描述了某地区汇总经济数据的处理流程。

阅读流程图可知，处理 1 的任务是输入基层单位上报的数据。为保证输入的数据的合法性，处理 1 还应对输入数据进行合法性检查。同时我们看到，指标体系文件参与了处理 1，这样将基层单位上报数据中的"指标名称"改为"指标代码"就成为可能，而且这样也可以为减少文件的存储量，使处理 2 和处理 3 能够更方便的汇总。

考查月基层数据的数据来源，它是由指标体系文件和基层单位上报数据经过处理 1 而成的，这样，我们就可以确定月基层数据的数据必须直接或间接从指标体系文件和基层单位上报数据中取得。而月基础指标数据经处理 2 生成的，从流程图来看，月基础指标数据的数据项必须全部直接从月基层数据取得，这样我们就可以确定月基础指标数据文件数据项的最大范围，即指标代码、指标名称、计量单位、单位名称和产值等 5 个数据项。考查上述 5 个数据项，显然，指标代码和指标名称的并存是一种存储浪费，考虑到文件长度和处理的简便性，我们保留指标代码；另外，由于流程图是用于地区的经济数据汇总，汇总后的数据是指整个地区的各指标的总产值，因此单位名称将不会参与以后的处理，可以删除。最后，基层单位上报的数据中只有产值，没有计量单位，由此我们可以推断对同一个指标各单位产值的计量单位都有是相同的，所以计量单位不必保存在月基层数据文件中，至于在输出经济数据月服表和分析报表时需要用到计量单位，则可以从指标体系文件中取得。

根据上述分析，月基础指标数据文件的记录只有指标代码和产值两个数据项。但要考虑到，基础指标数据文件中，对应于一个基指标码的记录只有一个，它存储汇总后的该基础指标的总产值。

指标代码的主要作用集中表现在以下 3 个方面：（1）指标码可用来代替指标名称，以减少文件的存储容量；（2）只要科学地设计指标代码，不但能描述各指标体系中的相互关系，而且由于指标代码明显比指标名称简单，便于处理，能够提高检索及处理速度；（2）当指标体系改变时，只须改指标体系文件，而不必须改程序，使程序与数据的相关性大大降低，提高了软件的可维护性。

【答案】

[问题 1]

（1）输入基层单位于报的数据，并进行合法性检查。

（2）用指标代码替换指标名称，形成月基层数据文件。

[问题 2]

指标代码，产值。

[问题 3]

（1）减少文件的存储容量；

（2）提高检索及处理速度；

（3）修改指标体系时只须修改指标体系文件，而不必修改理程序，便于维护。

试题 2（1995 试题 2）

阅读下列说明和流程图 6-27，回答问题 2，把解答写在答卷的对应栏内。

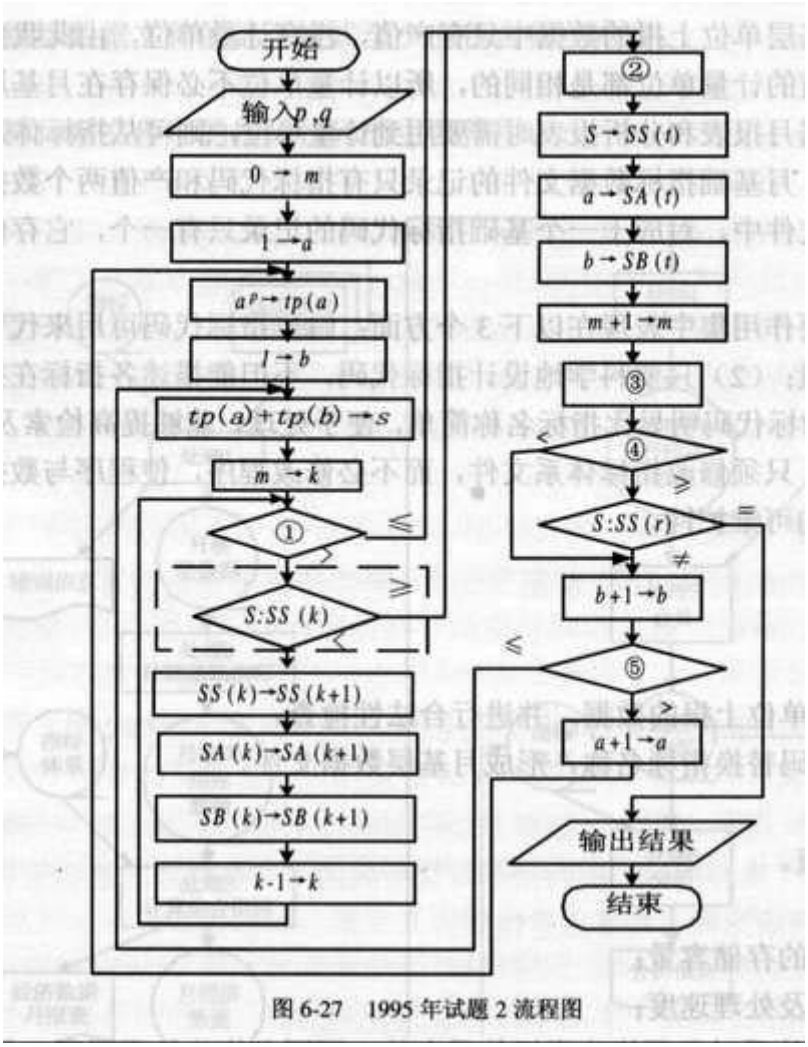


图 6-27 1995 年试题 2 流程图

[说明]

本流程图输入正整数 p 和 q（q≥2），寻找满足下列车员条件的 q 对自然数(a1b1),(a2b2),……(aq,bq)及最小的整数 sum。

1.ai ≥bI(I=1,2……q)

2.当 i≠j 时（aj,bj）(I=1,2…q)

3. $sum=a1^p+b1^p=a2^pb2^p=\cdots aq^p+bq^p$

例如：

当 p=2，q=2 时，$sum=50=7^2+1^2=5^2+5^2$;

当 p=3，q=2 时，$sum1729=12^3+1^3=10^3+9^3$;

当 p=3，q =3 时，$sum=8753919=426^3+167^3=423^3+228^3=414^3+225^3$。

本流程图采用枚举法，列举各种 aj^p、bj^p(a≥bi)及其和 $sum=aj^p+bj^p$，当发现 q 个相同的和时，即输出结果。图中，数组元素 $tp(K)=K^p$(k=1,2…)，枚举过程中产生的 sum 按递增顺序存放在数组 SS 中，相应的 ai 和 bI 存放在数组 SA 和 SB 中。

［问题 1］

填充流程图中的①～⑤，使之成为完整的流程图。

［问题 2］

若将流程图中的虚框部分改成图 6 -28,则流程图中的③、④应作怎样的修改。

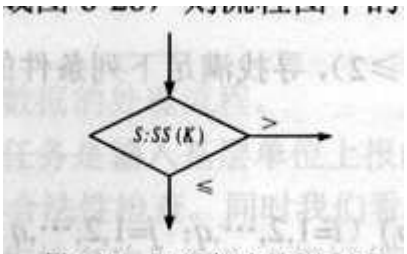


图 6-28 1995 年试题 2 问题 2

【解析】

该题用来寻找满足下列条件的 q 对自然数(a1,b1),(aq,bq)以及最小整数 sum：

1．ai＞= bI (i=1，2…，q)

2．当 I＜＞j 时，(aibi) (I=1,2,…q)

3. $sum=a1^p+b1^p=ap^q+bp^q$

仔细研究试题说明和流程图，我们可以确定流程图中各语句框的作用，并把流程图分解为 3 层循环；结合试题说明，仔细体会 3 重循环的功能，这里的关键是如何判断流程图中语句与试题说明的对应情况。

我们的思路沿流程图走，当走到填空①处时，不外乎有两种可能，即沿填空①的两个判断分支进行。很明显，如果我们沿"大于"分支前时，当执行 K-1→K 时将会出现数组越界问题，因为此前 K 的初值为 0。所以此处的判断该处寻找 S 在数组 SS 中合适位置的操作,则可知在两种情况下完成该操作,即 S 大于等于 SS[K]或 K 小于等于 0 时,这样,填空①的答案就非常明显了。那么填空②呢？②显然应该先给 t 赋初值，然后再在程序中使用，关键是赋什么初值，从上面的分析我们不难看出，S 应该插入 SS［K］的后面，这样我们就在②处填写 K+1→t。考查上述两个解答，它们能够完成在第一次循环时把处理导向②处的功能，应该是正确的解答。

我们且不论填空③和④及以后的语句是什么意思，有一点是可以肯定的，即在③和④中应该完成对 r 赋初值的操作，显然④是不能够胜任的,这样,我们应该在③中对 r 赋初值。流程图的剩余部分就是判别是否已有 q 个这样的 S （即 sum ）值。由于数组 SS 已按递增次序排列，所以具有相同的 S 值的元素一定在数组 SS 的连续存储区中。因为图中当 S＞=SS［K］时即将 S 插入，所以 S 的插入点 t 一定是该连续存储区的最后一个单元。如果存在 q 在相同的 S 值，则该连续存储区的第一个单元应在 t-q+1 处。所 r 的初值应该是 t-q+1，又由于在填空②中我们执行了 K+1→K 的操作，所以这里也可以赋初值 k-q+2。计算出 r 的值之后，显然，r 的值应该在区间［1，m］内。如果 SS［r］=SS，则认为找到一个解答，所以填空④的解答应该是 r:1。

结合试题说明中 aI 大于等于 bI 的提示,我们把结束中层循环的条件设置为 b:a，以便在 b 大于 a 时退出中层循环。

如果将图中虚线部分的判断框的出口条件改成=＜和＞，则仅当 S ＞SS（K）时才插入 S。此时 S 插在上述连续存区的第一个单元处。如果存在 q 个相同的 S，则该连续存储区的最后一个单元在 t+q-1 处。因此，应将填空③改成 t+q-1→r。由于 q＞0，所以 t+q-1=r＞=1 必定成立，故填空④应改成 m:r。

【答案】

［问题 1］

①K：0

②K+1→或 K+2-q→r

③t-q+1→或 K+2 -q→r

④r:1

⑤b:a

［问题 2］

③t+q-1→或 K+q→r

④m:r

试题 3（1995 年试题 3）

阅读下列说明和流程图,如图 6-29 所示,回答问题 1 和问题 2,把解答写在答卷的对应栏内。

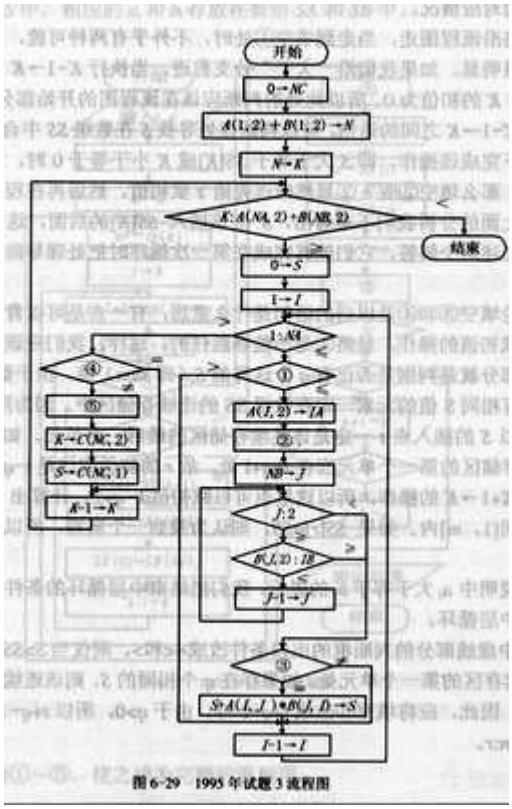


图 6-29 1995 年试题 3 流程图

［说明］

当一元多项式 $\sum_{i=1}^{n} a_i x^i$ 中有许多系数为零时，可用一个二维数组 D（M，2）来紧缩存储，其中 M 为多项式中非零系数的个数，数组的第一列存入非零系数的值，第二列存放该非零系数所对应的幂次，并且规定，数组元素按幂次的递减速次序存放。

例如，对应于多项式 $8X^5-2X^2+7$ 的二维数组内容如下所示：

$$\begin{pmatrix} 8 & 5 \\ -2 & 2 \\ 7 & 0 \end{pmatrix}$$

本流程图用来计算两个多项式的乘积，多项式的系数和幂次均按上述方式存放。数组 A、B 存放两个数欲相乘的多项式，它们的非零系数个数分别为 NA（＞0）和 NB（（＞0）），数组 A、B 存放两个数欲乘的多项式，它们的非零系数个数分别为 NA（＞0）和 NB（＞0），数组 C 存放结果（乘积）多项式，其非零系数个数用 ND 存储。

［问题 1］

填充流程图中的①～⑤，使之实现上述功能。

［问题 2］

若将流程图中 J：2 改成：J：1，则流程图能否正常工作，为什么？

【解析】

该题的流程图用来计算二个一元多项式的乘积。试题中建立一个二维数组，采用压缩存放的方式用来存放一元多项式，数组元索按幂次的递减次序存放，数组的第一列存放多项式中非零系数的值，第二列存放该非零系数所对应的幂次。题中还规定两个压缩存储的一元多项式 A 和 B 的乘积 C 也采用压缩存储。

分析流程图，该流程图由一个三重循环组成。外循环的循环变量为 K，初始值是 A（1，2）+B（1，2）。当 K 小于 A（NA，2）+B（NB，2）时外循环中 K 递减 1。由此我们推断，K 为 C 的某一个可能幂次。计算幂次为 K 的系数值，需要找出满足 A（1，2）+B（NB，2）＞K 时，满足 A（1，2）+B（J，2）=K 的 J 不可能存在，应该结束该循环，此时经 I+1→I 取下一个 A（I，2）。所以填空①应为 A（I，2）：K-B（NB，2）或 A（I，2）+B（NB，2）：K。当 A（I，2）≤K-B（NB，2）时先将 A（I，2）送给 IA，阅读流程图不难看出。内循环用来寻找值为 IB 的 B（J，2）。为使 A（I，2）+B（J，2）=K，IB 应为 K-A（I，2）。图中有语句 B（J，2）：IB，而此前没有对 IB 的赋值操作，所以填空②处应对 IB 赋值，至此，在填空②处填写 K-A（1，2）→IB 已经是非常明显的了。

由于内循环在 J＜或 B（1，2）≥IB 时结束，所以填空③应为 B（J，2）=IB 时才将 A（I，1）累加到 S 中。如此循环，直至 I＞NA，此时 S 中应该是对应于乘积的幂次为 K 的系数的值。考虑到经累加后的 S 值有可能为零，而 C 中只存储非零系数的值，所以填空④应为 S：0，这样只有在 S≠0 时才将其存储到 C 中。由于 NC 是非零系数的个数，在每一次将 S 存储到 S 中去之后，应该是对 NC 的递加，所以填空⑤处应为 NC+1→NC。

如果将流程图中的 J：2 改成 J：1，则当 J=1 时，B（J，2）：IB 的比较要继续进行。这样当 B（1，2）＜IB 时，J 的递减使 J=0，此时因 J＜1 而终止循环，必须造成③中 B（J，2）的下标 J 超出数组下标范围而发生数组越界。

【答案】

［问题 1］

①A（I，2）：K-（NB）或 A（I，2）或+B（NB，2）：K

②K-A（I，2）→IB 或 K-IA→IB

③B（J，2）：IB

④S：0

⑤NC+→NC

［问题 2］

不能，因为当 J=1 且 B（1，2）＜IB 时将因 J=0 而终止循环，从而导致③中 B（J，2）的下标超出

数组下标范围。

试题 4 （1995 年试题 4）

阅读下列说明和流程图，如图 6-30 所示，回答问题，把解答写在答卷的对应栏内。

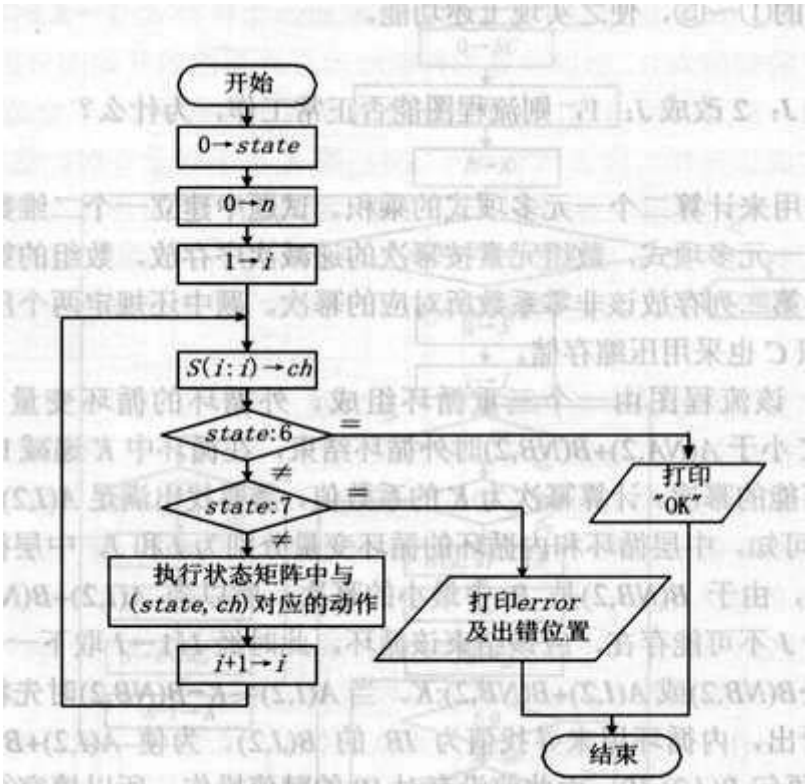


图 6-30 1995 年试题 4 流程图

［说明］
本流程图采用状态转换矩阵的方法来检验算术表达式（非空）的语法正确性，若发现错误，则指出发现错误的位置。

假定算术表达式中的运算对象仅由自然数及变量名（用标识符标识）组成，运算符均是双目运算符，有+ 、-、*、/**（由两个连续的"*"组成，表示乘幂运算）等 5 种，表达式中可以出现左右圆括号，并以字符#作为结尾。

状转换矩阵（见表 6-5）中的行代表当前状态（State），列代表读入字符，矩阵的内容（本题中只给出部分内容，空白部缺省）指出在当前状态下面临相应的读入字符时应执行的操作序列。

例如，若录前状态为 0 时面临的读入字符为″（″，则应执行的操作序例是″n+l→n:4→state″，即括号嵌套重数加 1，并把当前状态转为 4。

流程图中用字符串 S 存放被检验的算术表达式，用 S（I ：j）表示字串 S 中第 i 个字符至第 j 人字符（j≥i）的子串，其中 S(i:i)即为读入字符。State=5 表示退出一重括号嵌套；state=6 表示表达式语法正确；state=7 表示表达式语法出错。

［问题］

填充状态转换矩阵中的①～⑦ ，把相应的操作序列写在答卷的对应栏内。回答时可以使用如下形式的操作：

if 条件 then 操作 1 else 操作 2 ；
或 if 条件 then 操作；

【解析】

该题采用状态转换矩阵来检验非空算术表达式的语法正确性。

在该题的算术表达式中，可能出现的对象有自然数、变量名、运算符、左括号和括号。仔细阅读流程图可推知，开始状态为 0 状态。试题说明中还提到状态 6 表示被分析的表达式语法正确，状态 7 表示表达式语法出错，状态 5 表示退出一重括号的嵌套。

解答本题的关键是要明确各个状态的含义。阅读题中给出的流程图及状态转换矩阵可知，在 0 状态（开始状态）时遇到数字（表示表达式以自然数开头），则下一状态为 1，这样我们就可以断定 1 状表示前一个字符是自然数中的一位数字；在 0 状态时遇到字母（表示表达式以变量名开头），则下一状态为 2，这样我们就可以断定 2 状态表示前一字符是变量名中的一个符号；在 0 状态时遇到″（″（表示表达式以左括号开头），则将 n 加 1（从流程图可知，n 的初值为 0，n 加 1 表示进入一层括号），并使下一状态变为 4（即 4 状态表示前一字符是″(″）；在 1 状态时遇到″+″或-或″*″，或″1″，则下一状态为 3。即 3 状态表示前一个字符 运算时。据此，可以认为本题中的状态含义如下所示：

当前状态	前一个字符所属的对象
0	开始
1	自然数
2	变量名
3	运算符

4	（
5	）
6	正常结束
7	出错

有了以上的分析，只要我们能够对流程图和试题说明有充分的理解，解答①～⑥就非常容易了，兹不 述。只有⑦的解答应该仔细推敲。阅读状态转换矩阵可知，⑦是在当前状态为 3 时读入字符为"*" 且前一个字符也为"*"时才能组成乘幂次运算符，其他情况下不可能在当前状态为 3 时读入一个运算符，由此可以推断⑦时对"*"的处一。考虑到可能出现"***"这样的情况，所以⑦应为 ifs(I-1:i)="**"ands(I-2:I-2)＜＞"*"then3→state else7→state。

【答案】

①2→state

②2→state

③3→state

④7→state

⑤n-1→n;if n＞=0 then→state else7→state

⑥ifn=0 then6→state else 7→state

⑦ifs(I-I:I)="**"and (I-2:I-2)＜＞"*"then3→state7→state

试题 5 （1995 年试题 5）

阅读下列说明和流程图，如图 6-31 所示，回答问题，把解答写在答卷的对应栏内。

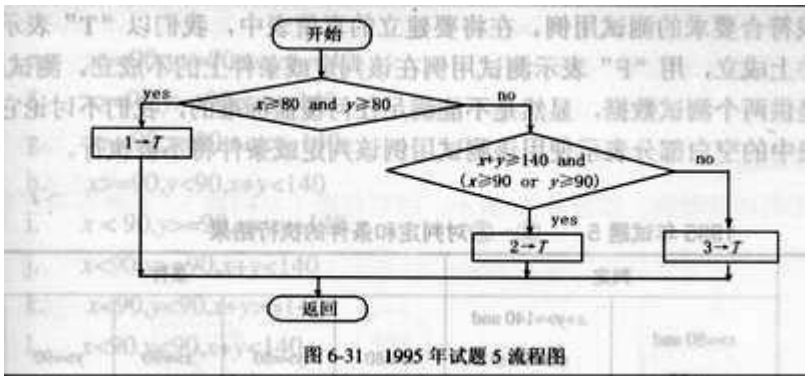


图 6-31 1995 年试题 5 流程图

［说明］

本流程图描述了某子程序的处理流程，现要求用白盒测试法对其进行测试。

［问题］

根据判定覆盖、条件覆盖、判定/条件覆盖、多重和条件覆盖（条件组合覆盖）和路径覆盖等 5 种覆盖标准，从供选择的答案中分别找出满足相应覆盖标准的最小的测试数据组（用①～⑩回答）。

供选择的答案

①x=90, y=90
②x=50, y=50
③x=90, y=90
x=50, y=50
④x=90, y=70
x=40, y=90
⑤x=90, y=90
x=50, y=50
x=90, y=70
⑥x=90, y=70
X=70, Y=90
x=50, y=50
⑦x=90, y=90
x=50, y=50
x=90, y=70
x=70, y=90
⑧x=90, y=90
x=50, y=50
x=90, y=50
x=80, y=80
⑨x=90, y=90
x=90, y=70
x=90, y=30
x=70, y=90
x=30, y=90
x=70, y=70
x=50, y=50
⑩x=90, y=90
x=80, y=80
x=90, y=70
x=70, y=90
x=30, y=90
x=30, y=90
x=70, y=70
x=50, y=50

【解析】

该题要求根据判定覆盖、条件覆盖、判定条件覆盖、多重条件覆盖和路径覆盖等盖标准，选择最小的测试数据组。

解答本题的关键是对 5 种覆盖标准有一个深入透彻的把握。由于此类试题在历年试题中经常出现，我们不妨在这里强调一下。判定覆盖是指选择足够的测试用例，使每个判定的所有可能结果至少出现一次。条件覆闰是指选择足够的测试用例，使判定中的每个条件的所有可能结果至少出现一次。条件覆盖是指选择足够的测试用例，使判定中每个条件的所有可能结果至少出现一次，并且每个判定本身的所有可能结构也至少出现一次。多重条件覆盖是指选择足够的测试用例，使每个判定中条件结果的所有可能组合至少出现一次。路径覆盖是指选择足够的测试用例，使流程图中的每条路径至少经过一次。

解答此类题目的简便方法是建立真值表（见表 6-6），通过测试用例在各个判定和条件上是否满足来寻找符合要求的测试用例。在将要建立的真值表中，我们以"T"表示测试用例在该判这或条件上成立，用"F"表示测试用例在该判定或条件上

的不成立。测试用例 1 和测试用例 2 仅提供两个测试数据，显然是不能满足任何覆盖标准的，我们不讨论它们满足哪种标准。真值表中的空白部分表示使用该测试用例或条件将不被执行。

表 6-6 1995 年试题 5 ①~⑧对判定和条件的执行结果

编号	判定		条件				
	x>=80 and y>=80	x+y>=140 and (x>=90 or y>=90)	x>=80	y>=80	x>=90	y>=90	x+y>=140
①	T		T	T			
	F	F	F	F	F	F	F
②	F	T	T	F	T	F	T
	F	F	F	T	F	T	F
③	T		T	T			
	F	F	F	F	F	F	F
	F	T	T	F	T	F	T
④	F	T	T	F	T	F	T
	F	T	F	T	F	T	T
	F	F	F	F	F	F	F
⑤	T		T	T			
	F	F	F	F	F	F	F
	F	T	T	F	T	F	T
⑥	F	T	T	F	T	F	T
	F	T	F	T	F	T	T
	F	F	F	F	F	F	F
⑦	T		T	T			
	F	F	F	F	F	F	F
	F	T	T	F	T	F	T
	F	T	F	T	F	T	T
⑧	T		T	T			
	F	F	F	F	F	F	F
	F	T	T	F	T	F	T
	T		T	T			

通过真值表，易知满足判定覆盖的测试用例为⑤，满足条件覆盖的测试用不着例为④，满足判定条件/覆盖的测试用例为⑦。至于路径覆盖，我们首先要考查流程图中共有几条路径，仔细阅读流程图可知，该流程图共有 3 条路径，分别是第一判定为 T、第一判定为 F 且第二判定为 T、第一判定为 F 且第二判定也为 F。从表 6-6 可知供选择答案⑤满足路径覆盖标准。

为分析满足多重条件覆盖标准的测试数据，我们先列出两个判定和各种条件组合情况。

第一判定：

a x＞=80, y=80
b x＞=80, y＜80
c. x＞=80, y＜80
d. x＞=80, y＜80

第二判定：

e: x＞=90, y＞=90, x+y＞=140
f. x＞=90, y＞=90, x+y＜140
g. x＞=90, y＞=90, x+y＞=140
h. x＞90, y＜90, x+y＜140
i. x＜90, y＞90, x+y=140

j. x＜90, y＞, x+y＜140
k. x＜90, y＜90, x+y＞=140
I. x＜90, y＜90, x+y＜140

因为当 x＞=90，y＞=90，x+y 不可能小于 140，所以 f 情况不可能出现。此外，当 x＞=90, y＞=90 时，流程图不会执行到第二判定，所以 e 情况也不可能出现。表 6-7 列出了供选择答案⑨覆盖上述二组条件组合的情况。显然，⑨满足多重条件覆盖标准。

表 6-7 1995 年试题 5 答案⑨

编号	测试数据	覆盖的条件组合
⑨	x=90,y=90	a
	x=90,y=70	b,g
	x=90,y=30	b,h
	x=70,y=90	c,i
	x=30,y=90	c,j
	x=70,y=70	d,k
	x=50,y=50	d,l

【答案】

判定覆盖 ⑤

条件覆盖 ④

判定/条件覆盖 ⑦

多重条件覆盖 ⑨

路径覆盖 ⑤

6.7 1994 年度软件设计试题解析

1994 年共有 5 道软件设计试题。试题 1 是软件分析试题，较多地涉及了流程图的评价与修改，这就要求考生在解答题目时将更多的注意力放在具体问题的解决上，仅有对给定流程图的理解还是不够的，关键是把握问题的实质，重在如何把问题的解决反映在流程图的评价与修改上。试题 2 和试题 3 也是难度适中的试题。试题 2 是从一个数组中查找一个特定的元素，试题说明中对算法有所交代，但关把握问题的实质，重在如何把问题的解决反映在流程图的评价与修改上。试题 2 和试题 3 也是难度适中的试题。试题 2 是从一个数组中查找一个特定的元素，试题说明中对算法有所交代，但关键部分语言不详，只能靠考生通过流程图来获得了。试题 3 是数据压缩变换的程序，难度较之试题 2 还要低些。试题 4 的考查形

成式比较特别，但难度还是较底。除了流程图修改部分以外，本年度的流程图填空试题总体来说难度较低，试题 5 年 SQL 语言查询比较容易。

试题 1（1994 年试题 5）

阅读下列说明和流程图，如图 6-32 所示，回答问题 1 至问题 3，将解答写在答卷的对应栏内。

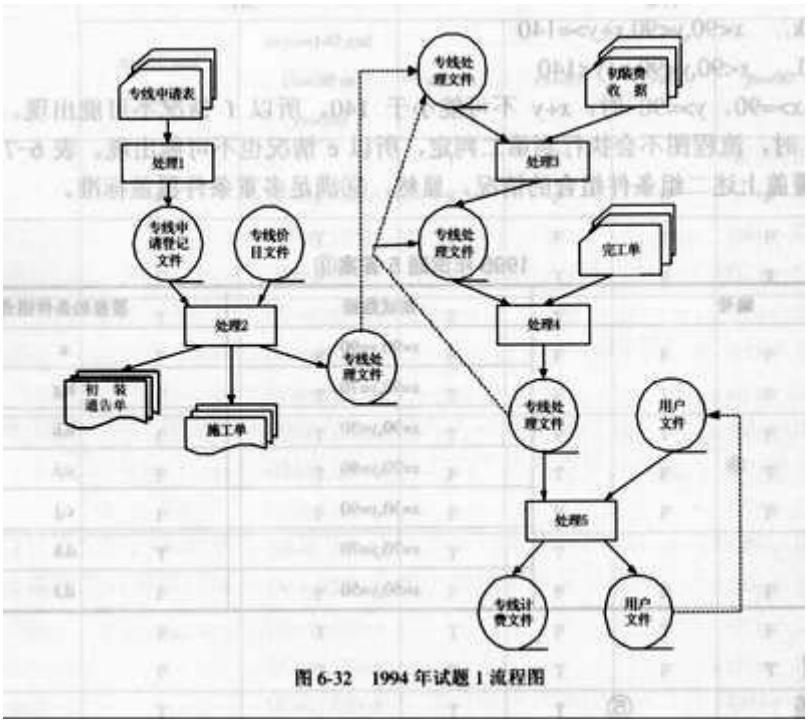


图 6-32 1994 年试题 1 流程图

［说明］

流程图描述了某电信局数据通信专结计费业务管理系统的部分处理流程。

1、凡申请专线者，均需填写专线申请表。系统把申请表存储在专线申请登记文件中，等待分配专线号。

2、系统为申请者指定专线号，并根据通信距离（按地区计算）、通信速率计算初装费和月租费，然后发初装通知单给用户，并产生施工单交有关部门施。同时产生专线处理文件。

3、施工结束后，系统更新用户文件，并产生专线计费文件，作为以后收费的依据。

4、一个用户可以租用多条专线，用户可用现金或银行托付两种方式支付租金，但一个用户只能使用一种付款方式。系统每月按用户（而不是专线）为单位计费出帐。

5、流程图中各数据或文件及有关单据所含的数据项如下。

专线申请表及专线申请登记文件：

申请号、用户名称、付款方式、开户银行代码、帐号、主端名称、主端地址、对端地址、对端所在地区、通信速率、设备接口、申请日期

专线处理文件：

申请号、专线号、用户名称、付款方式、开户银行代码、帐号、初装费、月租费、完工日期

初装费收据：

专线号、初装费、交费日期

施工单：

施工单号、专线号、主端名称、主端地址、对端地址、对端所在地区、通信速率、设备接口、申请日期、施工期限

完工单：

施工单号、专线号、完工日期

用户文件：

用户编号、用户名称、付款方式、开户银行代码、帐号

专线计费文件

专线号、用户编号、月租金、开通日期

［问题 1］

专线价目文件由哪些数据项组成。

［问题 2］

为了避免在用户尚未支付初装费时就去施工，有人提议将图中从处理 2 产生施工单改成从处理 3 产生施工单。试问从处理 3 能否产生施工单？为什么？

［问题 3］

当一个用户租用多条专线时，若允许该用户对其中的一些专线采用现金支付，对另一些专线采用银行托付方式，则在尽量减少数据冗余的前提下，应如何调整有关的数据文件。

【解析】

本题的流程图描述了某电信局数据通信的专线计费业务。

要解答问题 1，应该首先考查专线价目文件在流程图中参与了哪些处理，通过这些和理的输入数据和输出数据的对比来确定专线价目文件由哪些数据项组成。阅读流程图，确认专线价目文件只对与了处理 2。处理 2 的输入数据源是专线申请登记文件和专线价目文件，而输出数据流测是初装单通知、施工单和专线处理文件。结合试题说明，切实把握各文件的数据项组在。除初装通知单在试题说明中没有提到外，输出数据流中的初装费和月租费尚未明确数据来源，应该在专线价目文件中提供。应该注意到，初装费和月租费是根据通信距离和计算初装费和月租费是很困难的。很明显，对某个具体的电信局来说，只要明确了对端所在地区，就相当于明确了通信距离。这样左专线价目文件中只要设置对端所在地区、通信速率、初装费用和月租费 4 个数据项就可以保证系统的正常运行了。

对问题 2 的解答仍然需要考查处理的输入数据流和输出数据流。我们年，处理 3 的输入数据源是专线处理文件和初装费收据，它们数据项的并集中没有生成施工单所需要的主端名称、主端地址、对端地址、对端所在地区、通信速率、设备接口和申请日期等数据项，所以不能由处理 3 来生成施工单，除非由另外的数据文件参与处理 3。

本题的说明中规定，一个用户可以租用多条专线。用户可用现金或银行托付两种方式支付租金，但限制一个用户只能使用其中的一种付款方式。如要取消这一限制，允许租用多条专线的用户对具体的专线采用不同的付款方式，由此应该重新审视属性付款方式所在的实体，付款方式就不再是某个用户的属性，而是某条专线的属性。所以，应该在专线设计费文件中加上付款方式数据项，取消用户文件中的付款方式数据项。

【答案】

［问题 1］对端所在地区、通信速率、初装费、月租费

［问题 2］对端地址、对端所在地区、通信速率、设备接口等数据项。

［问题 3］在专线计费文件中加上付款方式数据项，取消用户文件中的付款方式数据项。

试题 2 （1994 年试验题 2）

阅读下列说明和流程图，如图 6-33 所示，回答问题 1 和问题 2，把解答写在答卷的对应栏内。

［说明］

流程图 6-33 用来实现在数组 A（n）中寻找第 K 大的元素。

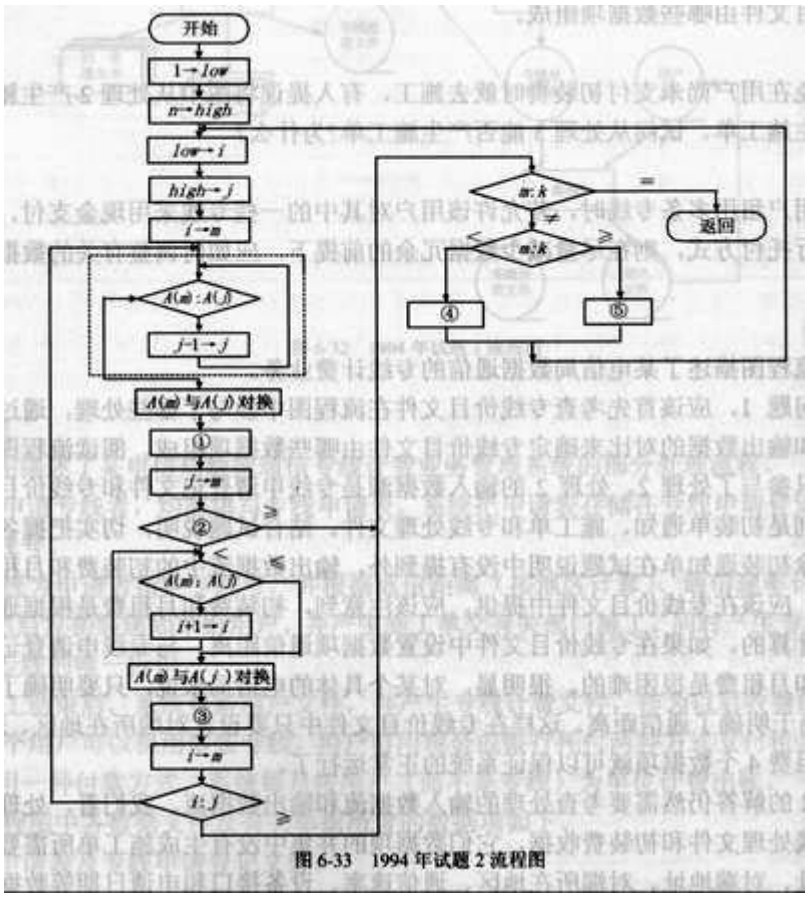


图 6-33 1994 年试题 2 流程图

算法思想是通过比较和交换，在区间 1～n 中找到一个 m，使 A(1)～A(m-1)的值均大于 A(m)，A(m+1)～A(n)的值均小于 A（m）即为第 K 大元素，否则调整查找区间，继续上述查找。

［问题 1］

流程图 6-33 中的虚像框部分改成图 6-34 所示的框图，则当 A（1）为数组 A 的最大元素时，修改后的流程图能否正常运行？为什么？

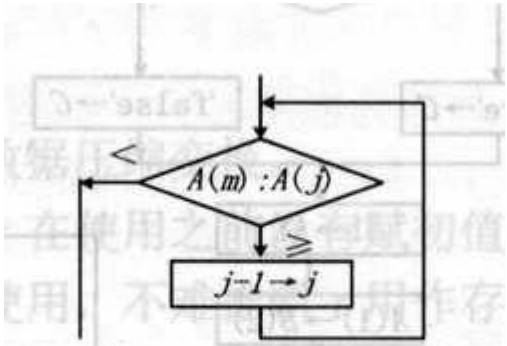


图 6-34 1994 年试题 2 问题 2

【解析】

本题的流程图用来寻找数组 A 中第 K 元素。

流程图中，首先给出了寻找的区间［low, high］，其初值为［i, n］。又给出两个指针 i 和 j，将初值分别不定义为 I=low，j=high，且 m=I。此后是一个循环，该循环完成在 A(m)大于等于 A(j) 时，A(m)与 A(j)的对调。让我们的思路沿流程图继续进行，填空①的作用是什么？我们且看①以后的语句，先是 m=j，然后是与上面基本相同的循环，只不过参与循环的对象已经是 A(i)与 A(m)。由于前面条件中有 A(i)≥A(j)，所以比较应从 i+1 开始，故填空①应为 i+1→I（或 m+1→）考虑到可能发生对换两个数组元素相邻时本次循环应该结束，即当 i≥j 时结束循环，填空②的解答应为 i:j。本次循环中是数组中靠前的元素依次与靠后的元素比较对换，在发生一次 A(i) 与 A(m)的对换之后。应该实现 j 的递减，然后执行 m→I，重复进行上述工作。显然填空③应为 j-1→j（或 m-1→j）。

如果对试题说明有充分的把握的话，应该能够用确填空④与⑤完成查找区间的调整。考查流程图，若 m=K,则第 K 大的数已经找到，程充结束。否则，若 m＜K，则下次的查找区间为［m+1,high］，所以填空④应为 m+1→low；若 m＞K，则下次的查找区间为［low m-1,］，所以填空⑤应为 m-1→high。

若将流程图中的虚框部分按题中问题 2 所述的方式修改，则当 A（1）为数组 A 的最大元素时，由于 A(1≥)A（j）（j=2，3…，n）所以修改后的流程图将始终执行 j-1→j，以至数组下标越界，因此修改后的流程图不能正常运行。

【答案】

［问题 1］①i+j→i 或 m+1→i

②i:j

③j-1→j 或 m-1→j

④m+1→low

⑤m-1→high

［问题 2］不能，因为当 A（1）为数组 A 最大元是修改后的流程图将始终执行 j-1→j，以至数组下标越界。

试题 3 （1994 年试题 3）

阅读下列说明和流程图，如图 6-35 所示，回答问题 1 和问题 2，把解答写在答案的对应栏内。

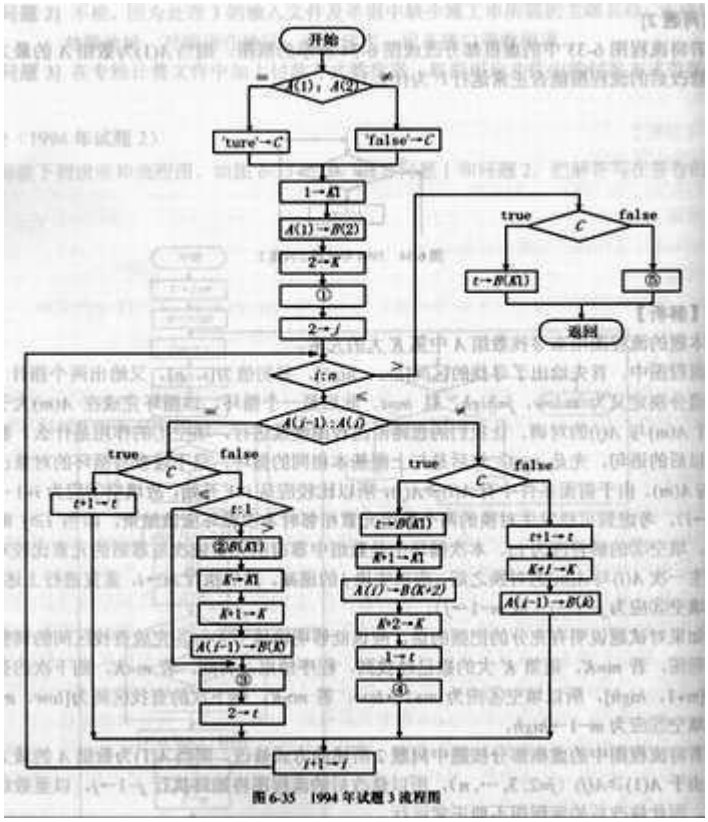


图 6-35 1994 年试题 3 流程图

［说明］

流程图 6-35 用来将数组 A 中的 n(n ≥2)个数经变换后存储到数组 B 中。变换规则如下：

1、若 A 中有连续 t 个相同的元素(t＞1)，则在 B 存入 t 和该元素的值；

2、若 A 连续 t 个元系(t≥1)，其中每个元素都与相邻的元素不相同，则在 B 中存入-t 和这 t 个元素的值。

例如：A={3，3，3，5，7，6，3，6，2，2，2，1，2}

则变换后 B={4，3，2，5，-4，7，6，3，6，4，2，-2，1，2}

流程图中，逻辑变量 C 用来区分正在进行连续相同元素的计数还是连续不等元的计数，KI 用来记录数组 B 存放 t 或-t 的元素的下标。

［问题 1］

填充流程图中的①～⑤，使之成为完整的流程图。

［问题 2］

如果删除流程图中的判断框 t:1，那么，当数组 A+{5，5，4，4}时，经改变后的流程图的变换，数组 B 将会有什么样的元素值？

【解析】

本题的流程图的功能是进行数据压缩变换。

仔细阅读流程图，发现变量 t 在使用之前没有赋初值，这样填空①的作用应该是给 t 赋实值。考查变量 t 在流程图中的使用，不难推断 t 用作存放连续相同元素个数或连续不同元素的个数，很明显，t 的初值就应该是 1，所以我们在①中填写 1→t。结合我们对流程图的理解，填空②所在分支处理的应该连续不等元素结束的情况，这是应该连续不等元素个数的负值赋给 B（K1），那么我们在②中填写-t 吗？如果仔细考查 t 在流程图中的作用，就可以推知这不是正确的答案，因为当前元素 A(I-1)已经不能包含在连续不等元素中了，所以我们在②中填写 1-t。

考查 C 的作用，易知当前处理的是连续相同的元素时，C 的值为 true；当前处理的是连续不等元素时，C 的值为 false。根据编程常识，这样的逻辑变量应该根据处理对象的不同而变反，如果我们敢于大胆推断，我们应该在填空③和④中填写将 C 变反的语句，即将 C 的值修改为进入该分支时 C 值的逻辑非。回过头来理解流程图，可以确认上述解答的正确性。
填空⑤与语句 1→B（K1）作用应该是相同的，由于最后一个元素页包括含在 t 的计数中， 我们已经不能参照②的解答思路，而是填写-t→B（K1）。

值得注意的是，当 A（i-1）≠A(i)且 C='true'时，虽然将'false'送给 C，但下一组并不一定是不同元。在问题 2 中给出的数组 A=（5，5，4，4）就是这种情况。当 i=3 时因原先 C='true'，现在 A（2）≠A（3），因此 1→t，'false'→C，且使 K1=3，K=4。然后 I 加 1 后变成 4，此时因 A（3）=A（4）且 C='false'，并且因 t=1 而执行'true'→C 和 2→t 如果取消流程图中的 t:1 框，那么就把前一组法作不同元来存入，此时 1≠A（3），因此 1-t'fase'→C 且使 KI=3K=4。然后 I 加 1 后变成 4，此时因 A（3）=A（4）且 C='false'并且因 t=1 而执行'true'→C 和 2→t 。如果取消程图中的 r;1 框，那么就把前一组当作不同元来存放，此时 1-t=0→B（3），4→Ki，A（3）→B（5 ），并且 K=5。当 i 加 1 变成 5 后，因 5＞n 所以执行 t→B（K1）那 2+B（K1）那 2+B（4）。因此当删除流程图中的判断框 t:1 后，该数组经变换后，B={2，5，0，2，4}。

【答案】

[问题 1] ①1→t ②1-t ③'true'→C ④'false'→C ⑤-t→B(K1)

[问题 2] B={2，5，0，2，4

试题 4 （1994 年试题 4）

阅读下列说明和流程图，如图 6-36 所示，回答问题 1 至问题 3，把解答在答卷的对应栏内。

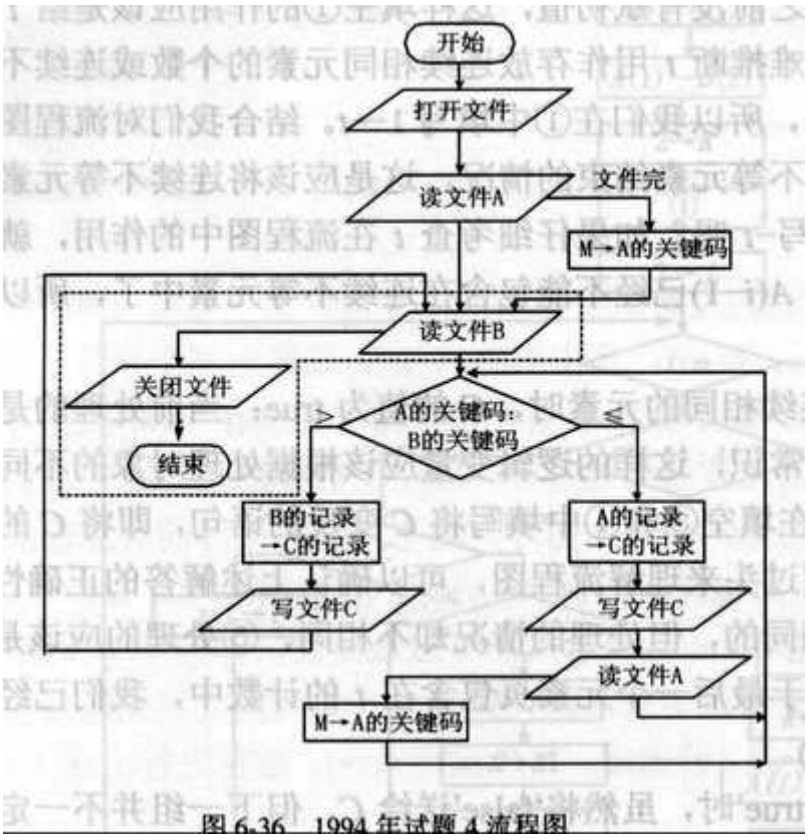


图 6-36 1994 年试题 4 流程图

[说明]

流程图 6-36 用于把文件 A 和文件 B 合并成按上升顺序分类（排序）的文件 C。已知文件 A 和 B 中的关键码均小于 M。

[问题 1]

为使流程图 6-36 能正确工作，文件 A 和文件 B 诸记录的关键码必须满足什么条件？

[问题 2]

按流程图 6-36 的处理方式，分别指出文件 A 和文件 B 至少应含有的记录个数。

［问题 3］

若将流程图 6-36 中的虚线部分改为框图 6-37，则图中的"判断条件"应是什么？

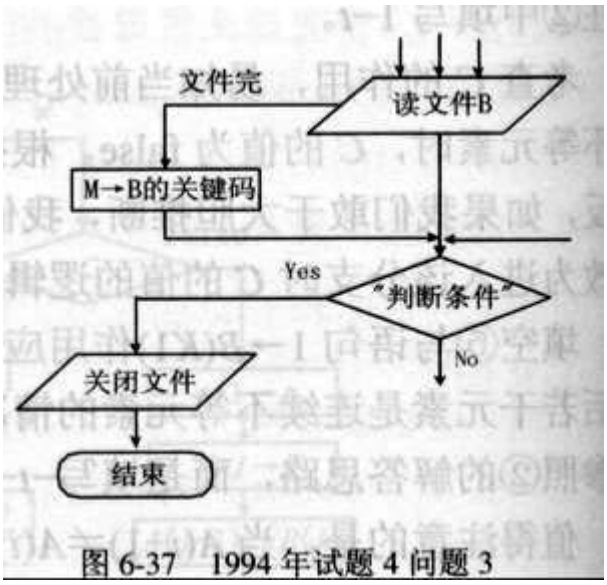


图 6-37 1994 年试题 4 问题 3

【解析】

本题的流程图用业实现两个文件的合并排序。

阅读流程图可知，图中逐个读取文件 A 记录和文件 B 的记录，当 A 记录的关键码大于 B 记录在关键码时，将 B 记录写入文件 C；反之，将 A 记录写入文件 C。显然，利用上述算法实现文 C 按升序排列，要求文件 A 和文件 B 在合并前必须已按升序排列，否则不能保证程序达到预期的要求。

流程图中，当文件 A 读完时，将 M 赋给 A 记录的关键码。结合试题说明可知，文件 A 和文件 B 中的关键码均小 M。所以，当赋给 A 记录的关键码后，A 的关键码总是大于 B 的关键码，使 B 记录依次写入文件 C，直至文件 B 读完。也就是说，当文件 A 先读完时能确保将文件 B 的剩余记录全部写入文件 C。但是，当文件 B 先读完时，就立即关闭文件，并结束运行。这样，当文件 B 先读完时，文件 A 的剩余记录无法写入 C 中。为达到将文件 A 和 B 的所有记录都合并到文件 C 中去的目的，本流程图要求必须先读完 A，后读完 B。这要求文件 A 的最后一个记录的关键码应该小于或等于文件 B 的最后一个记录的关键吗。

阅读流程图可知，当文件 A 为空时，流程图能正常运行（即使文件 B 也为空），运行的结果是文件 B 的所记录依次写入 C。然而，当文件 A 非空时，文件 B 必须非空，否则，文件 A 记录无法写入文件 C，因为文件 B 的空将导致程序的结束。因此文件 A 至少有零个记录；当文件 A 非空时，文件 B 至少有 1 个记录；当文件 A 为空时，文件 B 至少有零个记录。

根据上述分析可知。该流程图要求文件 A 的最后一个记录的关键码应小于或等于文件 B 的最后一个记录的关键吗，这人要求使程序对数据的要求非常苛刻，显然不是一个成功的程序。问题 3 将流程图作了改进，即当文件 B 读完时也将 M 赋给 B 记录的关键吗。这样不管哪个文件先读完，都能确保未读完的文件的剩余记录依次写入文件 C 中。由于将 M 赋给 A（或 B）记录的关键码的操作只能发生在文件 A（或 B）读完时，所以此时结束运行的判断条件应是两个文件都已经读完，即 A 的关键码等于 M 并且 B 的关键码等于 M。

【答案】

［问题 1］文件 A 和文件必须均按升序顺序，并且文件 A 的最后一个记录的关键码应小于或等于文件 B 的最后一个记录的关键吗。

［问题 2］文件 A 至少有个零个记录；当文件 A 非空时，文件 B 至少有 1 个记录；当文件 A 为空时，文件 B 至少有零个记录。

［问题 3］A 的关键码等于 M 并且 B 的关键吗等于 M。

试题 5 （1994 试题 5）

阅读下列说明和 E-R 图 6-38，回答问题，把解答写在答卷的对应栏内。

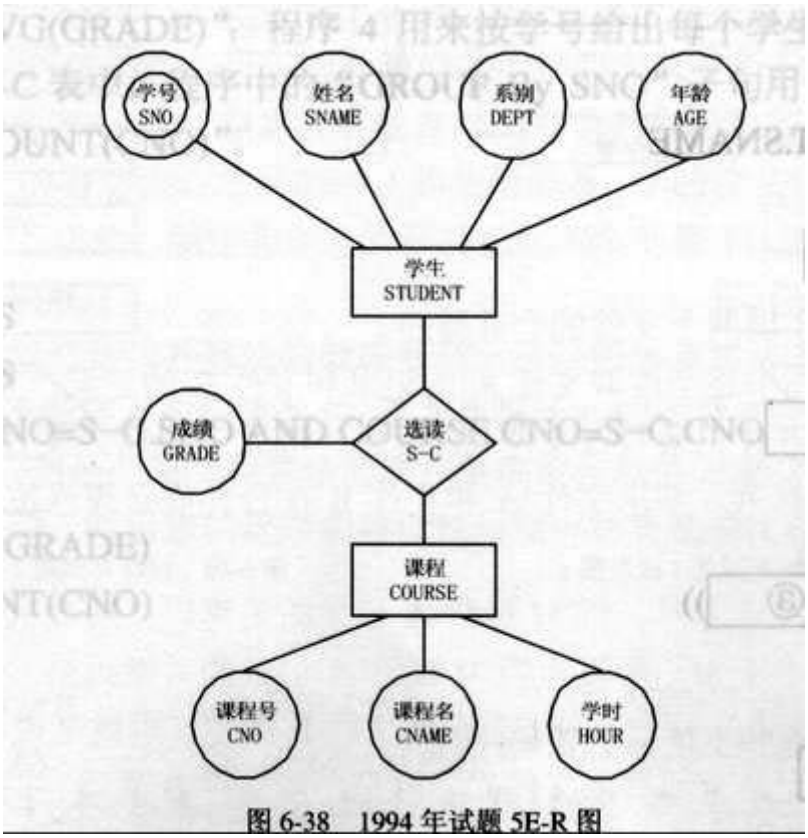


图 6-38 1994 年试题 5E-R 图

［说明］

设有下列关于学生成绩管理系统的 E-R 图。图中矩形表示实体，圆表示属性，双圆表示关键字属性，菱形表示实体间的联系。假定已通过下列 SQL 语言建立了基本表：

```
CREATETABLESTUDENT
（SNOCHAR（6）NOTNULL，
SNAMECHAT（20），
DEPTCHAR（20），
AGESMALLINT）；
CREATETABLECOURSE
（CNOCHAR6）NOTNULL，
CNAMECHAR（20），
HOUPSMALLINT）；
CREATETABLES-C
（SNOCHAR6），
CNOCHAR（6），
GRADESMALLINT）；
```

为了答题的方便，图 6-38 中的实体和属性时给出了中英两种名字，回答问题时只须写出英文名即可。

［问题］

填充下列 SQL 程序 1～4 中的①～⑥，使它们分别成以下查询功能。

程序 1：检索选读所有课程的学生姓名。

程序 2：给出全体学生人数。

程序 3：按学号给出每个学生的平均绩。

程序 4：按学号给出每个学生选读课程的门数。

［程序 1］

```
SELECTSTUDENT.SNAME
FROMSTUDENT
WHERE ①
（SELECT*
FROMCURSE
WHERE ②
（SELECT*
FROMS-C
WHERE ③
```

［程序 2］
SELECT ④
FROMSTUDENT

［程序 3］
SELECT ⑤
FRMS-C
GROUPBY SNO

［程序 4］
SELECT ⑥
FROMS-C
GROUPBY SNO
【解析】

本题采用 SQL 语言对基本表进行查询操作。解答此题的关键在于对 SQL 语言对基本表进行查询操作。解答此题的关键在于对 SQL 语句的透彻理解和灵活运用。从历年考试来看对 SELECT 语句的考果、查是重点，能否灵活地构造 SELECT 查询是提高此类题目得分率的关键。

程序 1 用来检索选读所有课程的学生姓名，程序 1 是一个嵌套的查询语句。最外层的 SELECT 是在 STUDENT 表中查找学生的的姓名，中层的 SELECT 在 COURSE 表中查找元组，最内层的 SELECT 在 S-C 表中查找元组。根据题目要求，找到的学生应"不存在这个/这些学生不学的课程"，即寻找那些中层 SELECT 的查找结果是空集的学生，所以填空①应为"NOTEXISTS"。同样，中层 SELECT 查找的课程应"是这个/这些学生不学的"，即寻找那些最内层 SELECT 的查找结果是空集的课程，所以填空②也是"NOTEXISTS"。填空③显然应是"STUDENT.SNO=S-C.SNO AND COURSE.CNO=S-C.CNO"。

程序 2 用来给出全体学生的人数，全体学生的人数就是 STUDENT 表中的元组个数，用 SQL 的内部函数 COUNT（*）即可得到，故填空④为"COUNT（*）"程序 3 用来按学号给出每个学生的平均成绩，学生成绩存放在 S-C 表中，程序中的"GROUPBySNO"子句用来按学号 SNO 分组，学号为 SNO 的学生的平均成绩可用 SQL 的内部函数 AVG（GRADE）计算，故填空⑤为"SNO，AVG（GRADE）"：程序 4 用来按学号给出每个学生选择读课的门数，学生的选课情况存放在 S-C 表中，程序中的"GROUP By SNO"子句用来按学号 SNO 分组，故填空⑥ "SNO，COUNTCNO"。

【答案】

①NOTEXISTS

②NOTESISTS

③STUDENT.SNO=S-C.SNO AND COURSE.CNO=S-C.CNO

④COUNT(*)

⑤SNO,AVG(GRADE)

⑥SNO,COUNT(CNO)

6.8 1993 年度软件设计试题解析

试题 1 提供了两种算法来生成一个数字序例，最后要求考生比较两种算法的优缺点。本题对考生的能力要求较高，而且由于提供了两种算法，解答试题的工作量也非常大，试题 2 竟然又是字符处理，这与 1992 年度试题 2 有雷同之处，突出了研究历年试题的重要性。试题 3 考查的是二叉树的非递增归遍历，这一直是一个难点，而且这里的二叉树又是一个顺序存储年度试题 4 大体相同，涵盖了数据文件、排序关键项和提高效率 3 个方面。另外几道试题涉及输出数字序列特定位置的数码、字符处理、SQL 语言、二叉树的非递归遍历等几个方面的问题。试题 5 是 SQL 语言，只要认真学习过数据库知识，解答本题还是比较容易。

试题 1（1993 年试题 1）

阅读下列说明和流程图，如图 6-39 和图 6-40 所示，回答问题 2，把解答写在答卷的对应栏内。

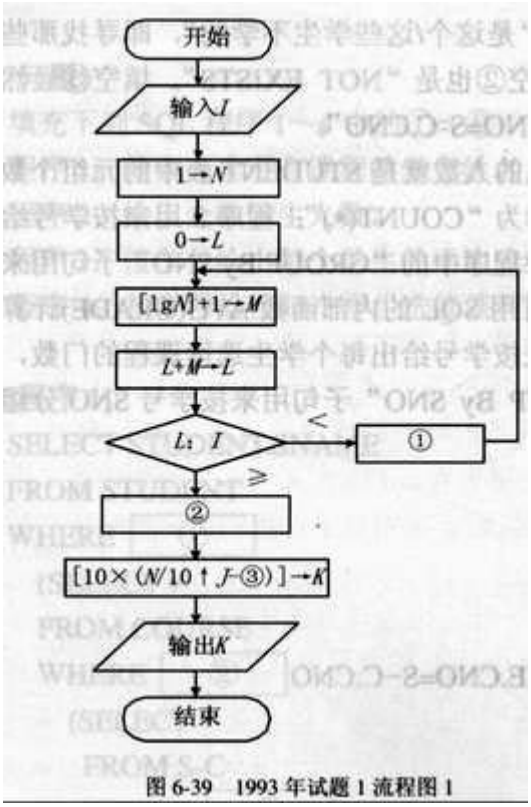


图 6-39 1993 年试题 1 流程图 1

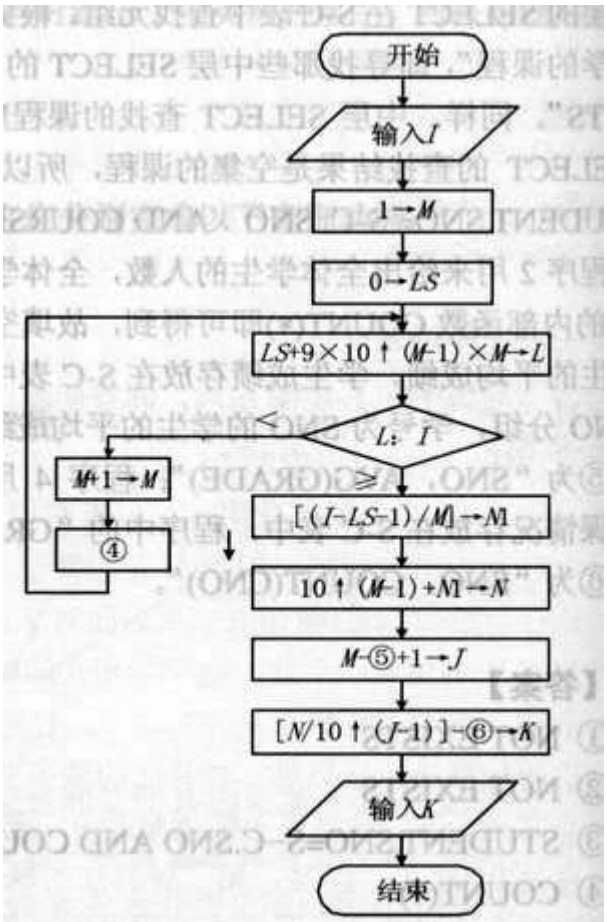


图 6-40 1993 年试题 1 流程图 2

［说明］

将自然数依次排列成如下所示的数码序列：

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16…

流程图 6-39 和流程图 6-40 都能输出从头数起的第 1 个数码。

流程图中 K 存输出数码，N 存放自然数，M 存放自然数的位数。图中表示乘幂运算，［W］表示不超过 W 的最大整数。

流程图 1（图 6-39）采用逐个增添自然数的方法。

流程图 2（图 6-40）采用一次增添位数相同的自然数序列段的方法。

［问题 1］填充流程图 1 和流程图 2 中的①～⑧ ，使之成为完整的流程图。

［问题 2］比较流程图 1 和流程图 2 的优缺点。

【解析】

本题提供了两种算法，从给定的数码序列中查找并输出第 I 个数码。仔细阅读试题说明中关于变量含义的说明，这对理解流程图有重要的作用。

先研究第一种算法。从流程图的开始部分往下阅读，结合对变量含义的理解，［lgN］+1 应该是自然数 N 的位数。从 L+M→来看，L 应该是到自然数 N 为止在序列中所占的位数。当 L 大于或等于 I 时，即表示 I 应该从此自然数中取得，所以我们在填空①中填写 N+1→N。接下来，填空②后面的语句使用了变量 J，但此前流程图中并没有给 J 赋值的操作，在程序中不经过赋值而直接使用量是不可思议的，由此可断定②是给 J 赋值的语句，但究竟赋什么值呢？这要结合上下文理解。从②的后续语句可知，J 表示序列中的第 I 个数码在 N 中的位数，所以我们在②中填写 L-I+I→J。为什么断定 J 表示序列中的第 I 个数码在 N 中位数呢？ 这要结合对填空③的解答来看。②的后续语句，［10x(…)］→K，就是说，10X（…）的结果是一个不大于 10 的实数，该实数的整数部分即为序列中的第 I 个数码。括号中得出的结果是一个小数，该小数的十分位为序列中的第 I 个数码，N/10J、③即为这个小数，由此印证 J 确实是序列中的第 I 个数码在 N 中的位数，并且了推断出填空③的解答应该是 N10J 的限整。

研究第二种算法，本算法采用一次将位数相同的自然数计入序列长度的方法。9×10（M-1）×M 实际上计算了位数为 M 的然数在序列中所占的长度，详细原因，不复详述，请读者自己理解。由此可以推断 LS 为已经计算的位数，L 为计入该段自然数后在序列中所占的总长度。由于 LS 需要循环使用，可以推断填空④的解答应该是 L→LS。当 L 大于等于 I 时，结合以上对流程图的理解，NI 应该是自然数 N 在当前自然数段中的位置，10（M-1）+NI 计算出应该在自然数 N 的确切值。从后面对 J 使用来看，J 的含义仍旧是序列中第 I 个数码在自然数 N 中的位置。至此，填空⑤和⑥的解答已经非常明显了，我们在这里给读者留一点思考的余地，不一一解释了。

分析两种算法，显然第二种算法的效率要高一些，因为该算法中的循环部分在实际执行中效率非常高。第 1 次循环 L 的增量是 9，第 2 次循环 L 的增量就达到了 180，第 3 次循环 L 的增量就是 2700 了。但是，第二种算法的缺点是确定自然数 N 和 N 中的第 J 位为序列所求的数位时算法明显复杂，写出来的程序的可读性与第一种算法相比相去甚远。

［答案］

［问题 1］①N+1→N ②L-I+1→J ③[N/10↑J] ④L→LS ⑤(I-LS-N1*M)
⑥[N/10↑J]×10

［问题 2]流程图 2 的处理效率比流程图 1 高，流程图 2 的算法比流程图 1 复杂。

试题 2 （1993 年试题 2）

阅读下列说明和流程图，如图 6-41 所示，回答问题 1 至问题 3，。把解答写在答卷的对应栏内。

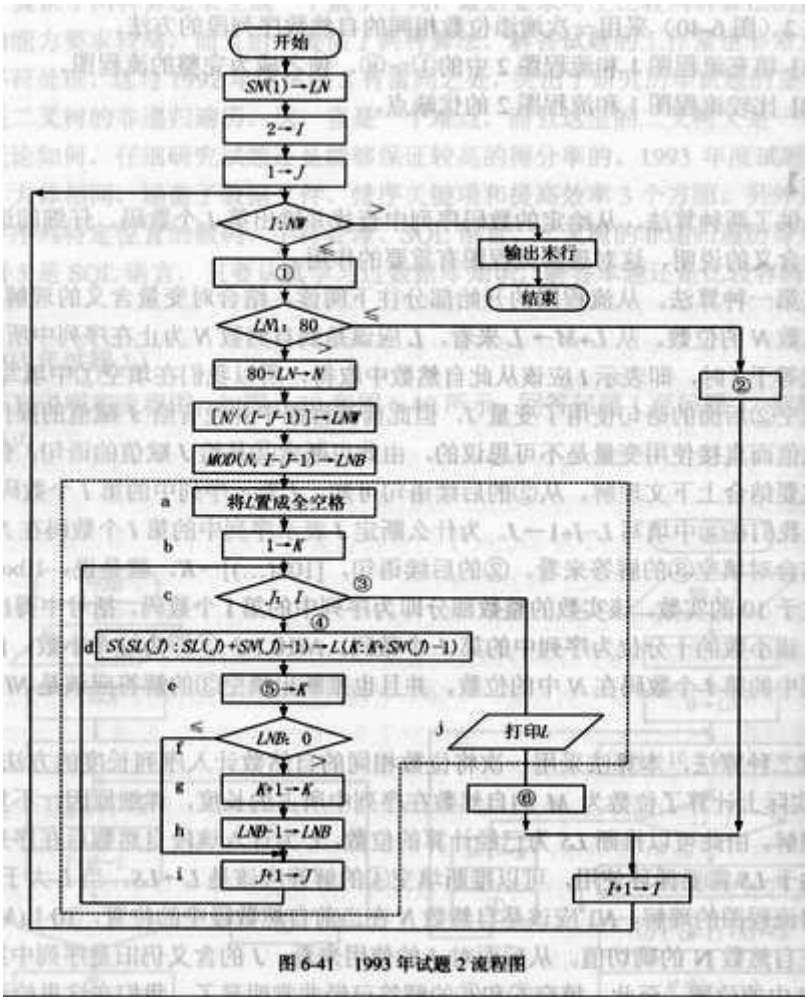


图 6-41 1993 年试题 2 流程图

［说明］ 本流程图的功能是对联预处理后的正文件进行排版输出。

假定预处理后的正文件存放在字符串 S 中，S 由连续的单词组成，单词由连续的英文字母组成。在预处理过程已产生以下信息：

变量 NW 存放正文中单词的个数，数组元素 SL（I）存放正文中第 I 个单词在 S 中的字符位置，SN（I）存放正文中第 I 个单词的长度。规定 S 中的字符位置从 1 开始计数，每个字符占一个位置。字符串 S 中的某个单词可用如下的子串形式来存取：

S（单词起始位置：单词终止位置）

并规定化在对字符串（或子串）赋值时，赋值号两端的字符串（或子串）长度必须相等。

排版输出的要求如下：

①每行输出 80 个字符;

②一个单词不能输出在两行中；

③除最后一行外，所有输出行既要左对又要右对齐。即每行的第一个字符必须是某个单词的第一个字母，最后一个字符必须是某个单词的最后一个字母；

④单词之间必须有 1 个或 1 个以上的空格；

⑤最后一个行只须左对齐，且单词之间均只有一个空格；

⑥使空格尽可能地均匀分布在单词之间，即同一行中相邻的单词的空格数量多相差 1。

假定正文中至少有两个以上的单词，每个单词的长度均小于 40。此外，流程图中省略了数据的输入部分。图中［W］表示不超过 W 的最大整数。

［问题 1］填充流程图中的①～⑥，使之成为完整的流程图。

［问题 2］图中的"输出末行"框未经细化，如果将图中的虚线部分复制到"打出末行"框上，那么复制部分应作怎样的修改？可用图中所标的 a、b……j 来回答，例如 a 改成 1→I ，删除 b。

［问题 3］ 如将图中开始部分的 SN（I）→LN 改成 0→LN，2→I 改成 1→I，则修改后的流程图是否正确。

【解析】

本流程图实现预处理后的正文排版输出。试题说明中给出了变量含义和排版的各种要求，我们应该形成如下的概念：流程图中使用了这些变量来实现符合要求的排版输出，各种要求都会在流程图中得到体现和实现，因此对变量说明和排版要求的切实理解就成为解答题目之前极为重要的工作。

阅读流程图，不难以现在①之后开始了对变量 LN1 的使用，由此推知①的作用是为 LN1 赋值，问题是赋什么样的值。考查下文，从 LN1：80 来看 LN1 应该是到此为此已经算在行中的单词和空格的长度（不能在最后一个单词后面缀上一个空格，这可以由试题说明的（3）推知）。那么此时的 LNI 究竟应该是什么呢？从流程图的开始部分仔细读下来，显然应该是 LN+SN（I）+1，这是符合要求的表示方式。解答了①，让思路顺着判断 LN1：80 小于等于的分支向前发展，LN1→LN 作为②的解答是顺理成章的事情，否则第二次运行到①处如何进行？

沿着 LN：80 的大于分支研读流程图。首先我们能够判断这是对单词总长度加间隔空格长度大于 80 后的处理。80-LN-N，这里的 LN 是尚未加入第 I 个单词时的行长度（包括间隔空格），因为加入第 I 个单词及该单词前的空格后行长将大于 80，所以不能将第 I 个单词包括在本行内。由此可知 N 的含义就是到第 I-1 个单词为止（包括间隔空格）本行剩余的位置（空格）。基于以上分析，以后的两句，先求将剩余的空格均匀分布在各单词的间隙里时每个间隙分担多少空格，后求均

匀分担后尚剩余多少空格无法均匀分布。由此我们可以推断 J 是本行第一个单词的序号。这样③和④的答案就很明显了。注意③填写大于等于是因为第 I 个单词不包含在本行内。不准理解 d 处的语句是取本行某个单词到 L 中，但⑤应该给 k 赋什么值？读到 J+1→J，我们才知道在 d 处之后需要计算下一个单词的超始位置。这里有 3 项工作要做，一是计入第 J 个单词的长度 SN（J），二是计入正常间隔空格的长度 L，三是计入每两个单词之间需要分担的均匀分布的空格数 LNW。现在这 3 项工作都需要在⑤中完成，所以⑤的答案应该是 K+SN（J）+1+LNW。

现在看⑥打印完一行后理所当然要打印下一行。在流程图的开始部分有 SN(1)→LN 所以这里的解答与之相似，应该是 SN(I)→LN ，以便开始新一行的处理，否则，LN 的值将停留在上一次 LN1→LN 时。注意这里的 LN 是上一行的最后一个单词，即第 I-1 个单词的长度，如果将该值带入下一行的处理，将会得到错误的结果。的处理，将会得到错误的结果。

处理未行要考虑的问题显然要少许多。f 处、g 处与 h 处是可以删除的，因为这里已经不存多余空格的均匀分布问题了，谈到这一点，⑤处应该改为 K+1+SN(J)。

如果按问题中的方案修改案修改流程图，结合①的解答，这将导致在统计第一行第一个单词时多加了一个 1，程序执行的结果当然也就是错误的。如果怀疑是在解答①时出错。那么可以试一试，这样的修改要改动多少地方才能实现试题说明所提出的各种要求。

[答案]

［问题 1］①LN+1+SN(I)→LN1 ②LN1→LN ③≥

④< ⑤K+1+LNW+SN(J) ⑥SN→LN

［问题 2］删去 f,g,h 框，将 e 改成 K+1+LNW+WN(J)→K。

［问题 3］不能。

［说明］

本流程图对顺序存储的二叉树按排非递归形式进行后序遍历打印。顺序存储的二叉树存放在数组 data（I:m）中，data 存放结点的值，right 存放指针值。

本题中顺序存储的二叉树是指对树中任意两个结点 node1 和 nde2（它们在顺序存储数组中的下标分别为 q1 和 q2），它们的指针值满足下列条件：

（1）如果子 node1 是根结点， 则 q1=1。如果从 node1 出发按前序遍历所得到的下一个结点是 node2，则 q2=q1+1。

（2）如果 node1 的左后件是 node2,则：

如果 node1 存在左后件，则 right(q1)=(q2+1)。

如果 node1 不存在左后件，则 right(q1)=-(q2+1)

（3）如果 node1 没有右后件，则 right（q1）=1;

如果 node1 存在左后件，则 right(q1)=;

如果 node1 不存在左后件，则当 node1 是按前序遍历的最后一个结点时，right(q1)=0，否则 right(q1)=1。

例如，下列二叉树的顺序存储情况见表 6-8。

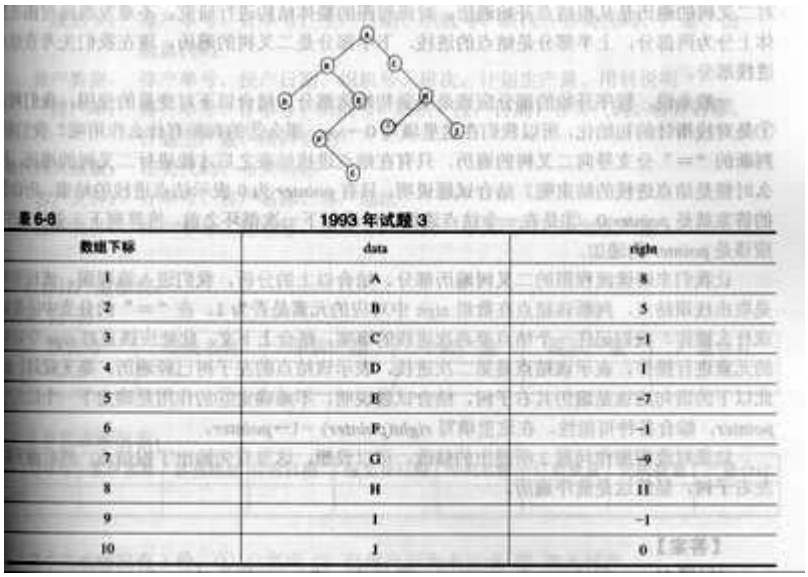


表 6-8 1993 年试题 3

数组下标	data	right
1	A	8
2	B	5
3	C	-1
4	D	1
5	E	-7
6	F	-1
7	G	-9
8	H	11
9	I	-1
10	J	0【答案】

流程图中 stak(1:N)是遍历过程中存放顺序存储数组下标的栈，sig(1:N)是配合栈操作设立的标志位（第一次进行栈时值为 1，第二次进栈时值为 2），变量 top 是栈顶指针，变量 first 是顺序存储二叉树的首指针。若树空，则 first=0，否则 first=1。指针 pointer 用来指出结点在数组中的的下标。

假定给出的顺序存储二叉树是正确的，且 stack 和 sign 都足够大， 不会溢出。

［问题 1］ 填充流程图中的①～⑤，如图 6-42 所示，使之成为整的流程图。

［问题 2］将流程图中的"打印 data(pointer)"框移至@处。则流程图执行的结果是什么？

【解析】

本题实现了一种按非递归形式对顺序存储的二叉树进行后序遍历输出的算法，仔细阅读试题说明，可以发现本题的难度不在于二叉树的后序遍历，而在于试题中提供的顺序存储方式，只有切实理解了了这种顺序存储方式，才能理解本题实现的后序遍历打印算法。

首先明确二叉树的前序遍历和后序遍历。前序遍历是指先输出根结占，再按前序遍历根结点的左子树，最后按前序遍历根结点的右子树。后序遍历是指先按后序遍历根结点的左子树，再按后序遍历根结点的右子树，最后输出根结点。我们知道，按非递归形式对二叉树进行遍历时需要栈的支持，一个结点在遍历过程中要两次进栈，第一次进栈是为了按后序遍历以该结点为根结点的左子树，第二次进栈是为了按后序遍历以该根结点的右子树。

首先我们要仔细推敲试题说明中的（1）、（2）和（3），对顺序存储的二叉树各结点的指针值的计算进行研究。然后阅读流程图，流程图的第一句就是将 first 赋给 pointer，这意味着对二叉树的遍历是从根结点开始遍历。对流程图的整体结构进行研究，不难发现流程图在整体上分为两部分，上半部分是结点的进栈，下半部分是二叉树的遍历。现在我们先考查结点进栈部分。

一般来说，程序开始的部分应该是变量初始化部分，结合以下对变量的使用，我们判定①是对栈指针的初始化，所以我们在这里填写 0→top。那么 ②的判断有什么作用呢？我们看，判断的"="分支导向二叉树的遍历，只有在结点栈结束之后才能进行二叉树的遍历。什么时候是结点进栈的结束呢？结合试题材说明，只有 pointer 为 0 表示结点进栈的结束，所以②的答案就是 pointer:0。③是在一个结点进栈之后进行下一次循环之前，推算剩下未做的工作，应该是 pointer 是递加。

让我们来研读流程图的二叉树遍历部分。结合以上的分析，我们进入流程图，流程图先是取出栈顶结点，判断该结点在数组 sign 中对应的元素是否为 1，在"="的分支中④要完成什么操作？我们记住一个结点要两次进栈的事实，结合上下文，此处应该是对 sign 中对应的元素进行操作，表示该结点是第二次进栈，表示结点的左子树已经遍历。毫无凝问，由此以下的语句应该是遍历其右子树，结合试验题说明，不难确定⑤的作用是确定下一个结点的 poiner，综合各种可能性，在这里填写 right(pointer)-1-pointer。

如果对流程图作问题 2 所提出的修改，可以设想， 这里首先输出了根结点，然后遍历其左右子树，显然这是前序遍历。

[答案]

［问题 1］①0→top ②pointer:0 ③pointer+1→pointer

④0→sign(top) ⑤right(pointer)-1→pointer

［问题 2］按前序遍历。

试题 4 （1993 年试题 4）

阅读下列说明和流程图，如图 6-43 所示，回答问题 1 至问题 3，把解答写在答卷的对应栏内。

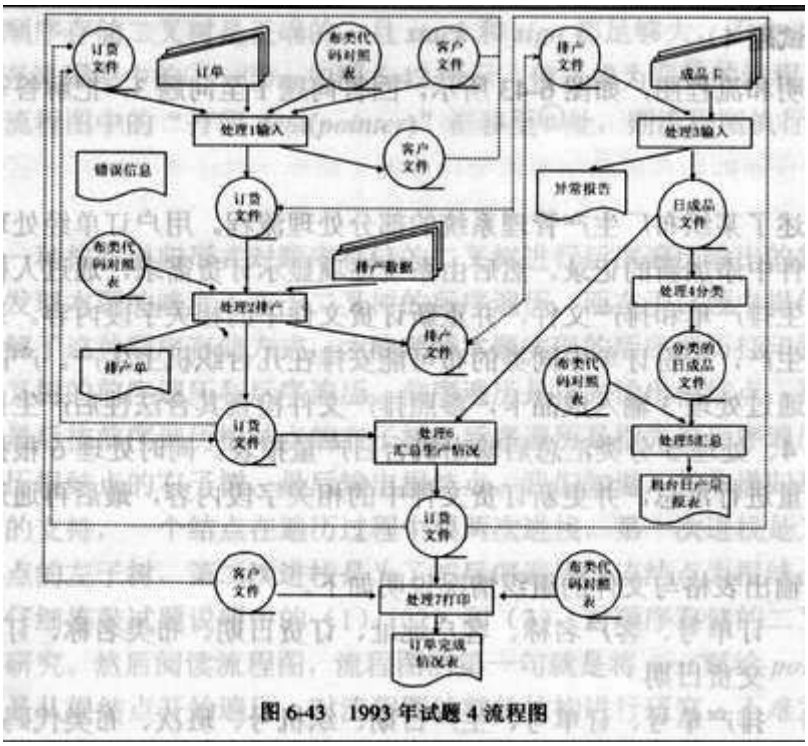


图 6-43 1993 年试题 4 流程图

［说明］

本流程图描述了某织布厂生产管理系统的部分处理流程。用户订单经处理 1 输入，在订货文件和客户文件中添加新的记录。然后由系统逐项显示订货需求，通过人机交互在处理 2 中进行排产，阁下排产单和排产文件，并重新订货文件中的相关字段内容。系统产生的排产单交给值机人员生产，一份订单中同类的布可能安排在几台织机上生产。产品合格后填写成品卡。系统每天能过处理 3 输入成品卡、参照排产文件检查其合法性后产生日成品文件。日成品文件经处得 4、处理 5 分类汇总后输出机台日产量报表。

输入单据、输出表格与文件的组织情况说明如下。

订单：订单号、客户名称、客户地址、订货日期、布类名称、
订货数量、应交货日期

成品卡：排产单号、投产日期、织机布、班次、布类代码、产量、
值机员代码

排产数据：排产单号、投产日期、织机号、班次、计划生产量、
用料说明

排产单：排产单号、订单号、织机号、班次、投产日期、布类代码、
布类名称、计划生产量、用科说明

布类代码对照表：布类代码、布类名称

客户文件：订单号、客户名称、客户地址

排产文件：排产单号、订单号、织机号、班次、布类代码、计划生产量

机台是产量报表：

年 月 日

织机号	布类名称	早班产量	中班产量	晚班产量	产量合计

订单完成情况表：

订单号	客户名称	客房地址	订货日期	应交货日期	布类名称	订货数量	完成数量	产量合计

其中完成情况有 3 种：①已完成 ②已排产但俚尚未完成 ③尚未财产。

［问题 1］分别指出订货文件和日成品文件至少应包括哪些数据项。

［问题 2］指出处理 4 分类的第一、第二关键项。

［问题 3］为提高处理 6 的效率，流程图（如图 6-43 所示）需作何补充？（用文字简述）

【解析】

本题描述了某织布厂生成管理系统的部分处理流程。仔细考查试题说明和流程图，发现在阅读流程图时要注意订货文件和日成品文件在系统中的数据来源和数据流向，还要注意处一 4 和处理 6 处理的特点。

定货文件的主要数据来源是订单，考查布类代码对照表和客户文件，为减少存储和处理订单文件的系统开销，订单中和客户名称和客户地址在以后的处理中可以通过订单号从客户文件中取得。布类名称 可以简人继布类代码，在以后的处理中通过布类代码对照表取得。仔细向后阅读，在处理 7 通过订单文件产生订单完成情况表，订单中的完成数量和完成情况尚未在系统中出现，我们怀疑它来自日成品文件，但处理 6 汇总生产情况时就该是将日成品文件中的苛些数据写入订单文件，然后由订单文件独自参国处理 7，这样，我们就把完成数量和完成怀况两个数据项加入订单文件。

考查是日产文件，发现日成品文件的主要作用不是参与处理 6，而是流程图的右半部分，经处理 4 和处理 5 最后生成机台日产量报表。机台日产量报表中的数据必须能够直接或间接从日成品文件取得。由于日成品文件是参与处理 6 的，如果该文件中没有数据项订单号，则在处理 6 中难以实现完成数量和完成情况的汇

总；由于处理 5 有布类代码对照表的参与，在日成品文件的中尽可以使用布类代码而不是布类名称。此外，机台日产量报表中有早班产量、中班产是、晚班产量和产量合计 4 个产量数据，我们可以在日成品文件中加入班次和产是两个数据项来支持以上数据需求。这样，包括日成品文件中不可缺少的织机号，日成品文件中就应该包括含订单号、织机号、布类代码、班次和产量 5 个数据项。

考查处理 4 的和理对象日成品文件和分类的日成品文件经处理 5 生成机台日产量报表等事实，如果处理 4 中分类关键项设置合理的意义就在于提高处理 5 的效率。由机台日产量报表的数据结构可知 ，知机号和布类名称的有序性对处理 5 的效率由很大影响，所以我们断定处理 4 分类的第一关键项是织机呈，第二关键项是布类代码。

考虑处理 6 进行汇总处理，提高处理 6 的效率的关键在于处理对象的有序性，并且两个处理对象是否经过按相同的关键项进行的有充化处理。考查日成品文件和订单文件，发现它们都可以近订单和布类代码进行有序化处理，由于处理 6 的主要功能是对订货文件进行处理，所以如果订货文件和日成品文件在通过处理 6 时先按订单号（第一关键项）和布类代码（第二关键项）进行有序化处理，处理 6 的处理效率必将大大提高。

【答案】

［问题 1］ 订货文件：订单号、订货日期、应交货日期、布类代码、订货数量、完成数量、完成情况。

日成品文件：订单号、织机号、布类代码、班次、产量

［问题 2］第一关键项是织机号，第二关键项是布类代码。

［问题 3］在处理 6 前分别对订货文件和日成品文件按相同的关键项（订单号和布类代码）分类。

试题 5 （1993 的试题 5）

阅读下列说明和 E-R 图 6-44，回答问题 1 至问题 3。

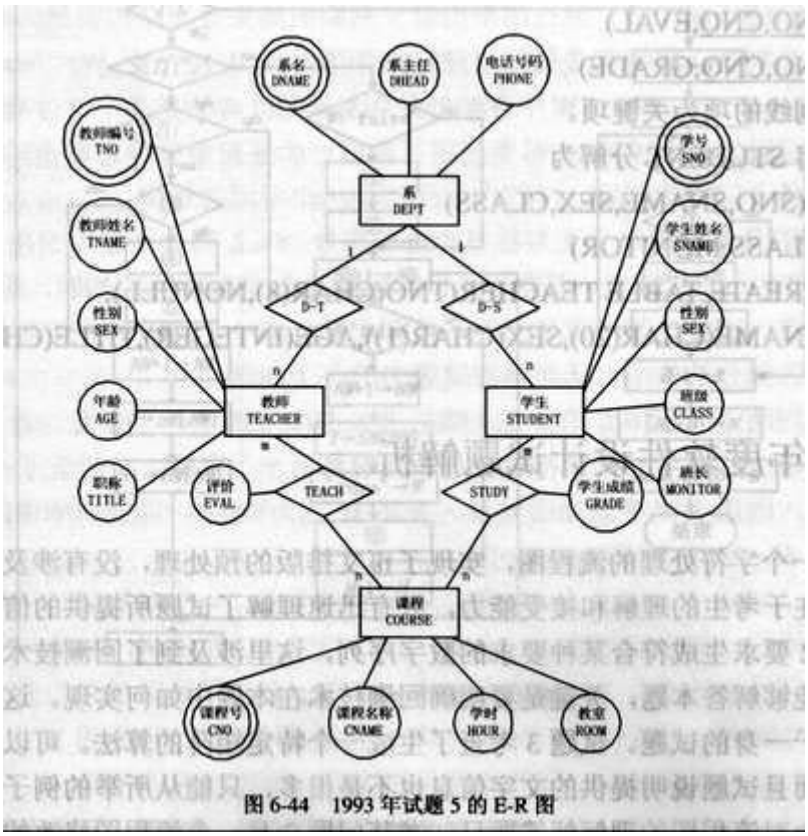


图 6-44 1993 年试题 5 的 E-R 图

[说明]设有下列关于教务管理系统的 E-R 图。图中矩形表示实体，圆表示属性，双圆表示关键字属性，菱形表示实体间的联系。为了答题的方便，图中的实体和属性同时给出院中英文两种名字，回答问题时只须写出英文名即可。

[问题 1]写出与上述 E-R 图对应的关系模式，并且用下划线标明相应的关键字。

[问题 2]问题 1 中的关系模式属于第几范式？如果属于第三范式，则说明理由；如果不属于第三范式，同将它化为第三范式（回答时只需写出修改的部分）。

[问题 3] 试用 SQL 语言定义教师（TEACHER）模式。回答时字段的数据类型以及题中未指明的名字由考生自己定义。

【解析】

本题不作详细解析，有关数据库的内容经常在下午试题中出现 。只要对有关数据库的理论知识和 SQL 语言有切实的把握，解答此类题目的成功率应该是很高的。建立 E-R 模型的基本步骤是，确定实体类型和联系类型，据此画出 E-R 图，然后确定实体类型和联系类型的属性，其中联系类型至少应包括与之联系的所有实体类型的关键项。

【答案】

[问题 1]

DPT（DNAME，DHEAD，PHNE）

TEACHER（TNO，TNAME，SEX，AGE，TITE）
STUDENT（SNO，SUAME，SEX，CLASS，MONITOR）
COURSE（CNO，CNAME，HOUR，ROOM）
D-T（DNAME，TNO）
D-S（DNAME，TNO）
TEACH（TNO，CNO，EVAL）
STUDY（SNO，CNO，GRADE）
其中有下划线的项为关键项。

［问题 2］将 STUDENT 分解为
STUDENT（SNO，SUAME，SEX，CLASS）
SCLASS（CLASS，MONITOR）

［问题 3］CREATETABLETEACHER（TNO（CHAR（8），NONULL），
TNAME（CHAR（20），SEX（CHAR（1））AGE（ENTEGER），TITLE（CHAR（20））

6.9 1992 年度软件设计试题解析

试题 1 是一个字符处理的流程图，实现了正文排版的预处理，没有涉及固定的算法，解答问题的关键在于考生的理解和接受能力，只有迅速理解了解试题所提供的信息才能够正确解答问题。试题 2 要求生成符合苛种要求的数字序列，涉及到了回溯技术，但并不是掌握了回溯技术就能够解答本题，关键是要明确回溯技术在本题中如何实现。这是一道集知识考查与能力考查于一身的试题。试题 3 考查了生成一个特定矩阵的算法，可以说，本题没有涉及固定算法，而且试题说明提供的文字信息也不是很多，只能从听举的例子中体会算法要实现的结果，结合对流程图的理解解答题目，尤其问题 2 是一个流程图修改的问题，如果没有对流程图的透彻把握，解答这个问题要大费周折。

试题 1（1992 年试题 1）

阅读下列说明和流程图，如图 6-45 所示，回答问题 1 和 2，把解答填入答卷的对应栏内。

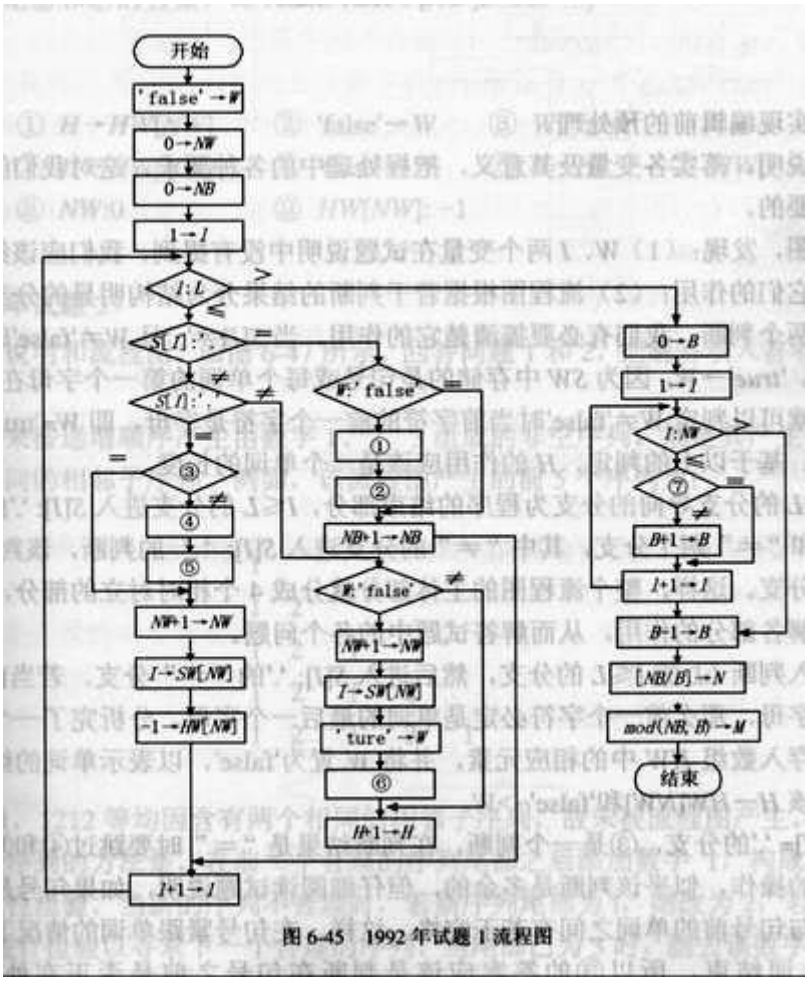


图 6-45 1992 年试题 1 流程图

[说明]

在字符数组 S 中存放着一行和度为 L 的正文，每个数组元素存放一个字符。现假定正文仅由单词、空格和句号组成，单词组由连续的英文字母组成。单词与单词之间可以有 1 个或多个空格，单词至名号之间或句至单之间或名号至单词之间可以有 0 个或 0 个以上的空格，两个名号之间除空格符外至少有一个单词。S 的第一个非空格字符不能是句号，最后一个非空格字符不一定是句号，S 的两端可以有 0 个或 0 个以上的空格。现准备对 S 中的字符串进行编辑，使得空格尽可能均匀地分布在单词这间。

该流程图实现编辑前的预处理。在流程图中，首先统计 S 中单词和句号的总数 NW、空格总数 NB、句号或每个单词的第一个字母在 S 中的位置（存放在数组 SW 中）和每个单词或句号的长度（存放在数组 HW 中，句号的长度置为-1）。然后，计算单词以及句号至单词间的间隔总数 B，最后计算每个间隔的平均空格数 N(整数）和剩余的空格数 M。

[问题 1]

填充流程图的①～⑦框，使之成为完整的流程图。

[问题 2]

为了检查正文中句号的用法是否正确，需要在流程图的（i）处增设判断及报错处理（见图 6-46）。试用题中的有关符号填写判断框⑧和⑨的内容。

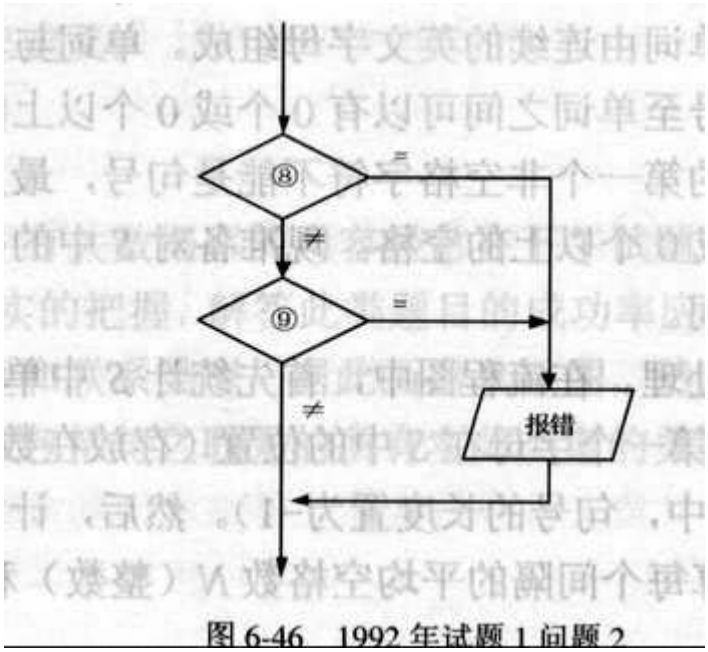


图 6-46 1992 年试题 1 问题 2

【解析】

该流程图实现编辑前的预处理。

阅读试题说明，落实各变量及其意义，把握处理中的种要求，这对我们阅读和理解流程图是极为重要的。

粗读流程图，发现：（1）W、I 两个变量在试题说明中没有提到，我们应该结合对流程图的理解来确定它们的作用；（2）流程图根据若干判断的结果分为结构明显的分支。

W 关系到两个判断，我们有必要搞清楚它的作用。当 且，时，句号数加 ，因为 SW 中存储的是句号或每个单词第的第一个字母在 S 中位置，这样，基本上就可以判定 W 时当前字符有前一个字符是字母，即 W 时程序正在职分析某个单词，基于以上的判定，H 的作用应该是一个单词的长度。

I：L 的 I L 的分支导向的分支为程充的结束部分， 的分支进入 的判断，该判断生″=″和″≠″两个分支进入 ，的判断，该判断产生″=″和″≠″两个分支。这样，整个流程图的主体部分就分成 4 个相对对立的部分，我们可以结合试题说明理解各部分的作用，从而解答试题中的各个问题。
我们先进入判断 I：L 的 I≤L 的分支，然后进入 的″=″分支。若前字符是’’且前一个字符是字母，那么前一个字符必定是单词的最后一个字母。分析完了一个单

词，应该将当次的长度存入数组 HW 的相元素，并将 W 置为'false '，以表示单词的结束，所以①和②的答案应该 H HW NW 和 falseW。

再进入 S 的分支，是一个判断，在判断结果"="时要跳过④和⑤，直接执行以下处理句号的操作，似乎该判断是多余的，但仔细阅读试验题说不得明，如果句号是可以紧跟单词的，也可以与句号前的单词之间有若干空格。这样，在句号紧跟单词的情况下，我们还应该处理一个单词结束，所以③是一个判断，在判断结果"="时要跳过④和⑤，直接执行以下处理句号的操作，似乎该判断是多余的，但仔细阅读试题说明，如果句号是可以紧跟单词的，也可以与句号前的单词之间有若干空格。这样，在句号紧跟单词的情况下，我们还应该处理一个单词结束，所以③的答案应该是判断在句号之前是否正在处理一个单词。④和⑤的处理与①、②的处理是相同的，答案也应该相同。

现在进入 的分支，若当前字符不是句号且 W ，这是在处理一个单词，只要在单词的长度上加 1 即可，由此可以推知 H 的作用是记录单词的长度；若当前字符不是句号有 W='false'，说说当前字符是单词的第一个字母，此时应该进行的操作是可以推断而知的，结合⑥后 H+1-H， 我们在⑥处填写 0H。

I＞L 的分支是统计完成以后的工作，即试题说明中的最后两句。各变量的意义已经明确，流程图的任务也经明确。从流程图的最后 3 句可以看出，B 应该是正文单词数减 1。如何得出 B，结合我们对流程图的理解，只有对数组 SW 或 HW 进行统计。两个数组中都有关于句号的记录，且在 SW 中句号的记录与单词起始位置的记录无区别的，但在数组 HW 中句号的记录却有一个特殊值。综合以上分析，我们在⑦中填写"HW -1"。

题中的问题 2 要求在流程图的 I 处增设判断及报错处理。仔细阅读处理说明中关于处理的要求，如下两点应该注意：（1）正文的第一个非空字符不能是句号；（2）且两个句号之间除空格外至少有一个单词。我们解答应该以这两点为基础来展。阅读流程图可知 ，句号在数组 HW 中的记录为-1，它在数组 HW 的位置由 NW 指定，NW 初值为 0。所以，当遇到句号，且 NW=0 时，表示该句号是正文的第一个非空字符。若 NW≠0,但 HW =-1，MJ 则表示当前的句号之前，除空格外也是句号，以上的分析函盖了我们提出的两点，用试题中的符号表示就是⑧和⑨的答案，即 NW：0 和 HW［NW］-1。

【答案】

［问题 1］①H→HW［NW］ ②'false' →W ③ W：'false'
④H→HW［NW］ ⑤'false' →W ⑥0→H ⑦HW

［问题 2］⑧NW：0 ⑨HW［NW］：-1

试题 2 （1992 年试验题 2）

阅读下列说明和流程图，如图 6-47 所示，回答问题 1 和 2，把解答案填入答卷的对应栏内。

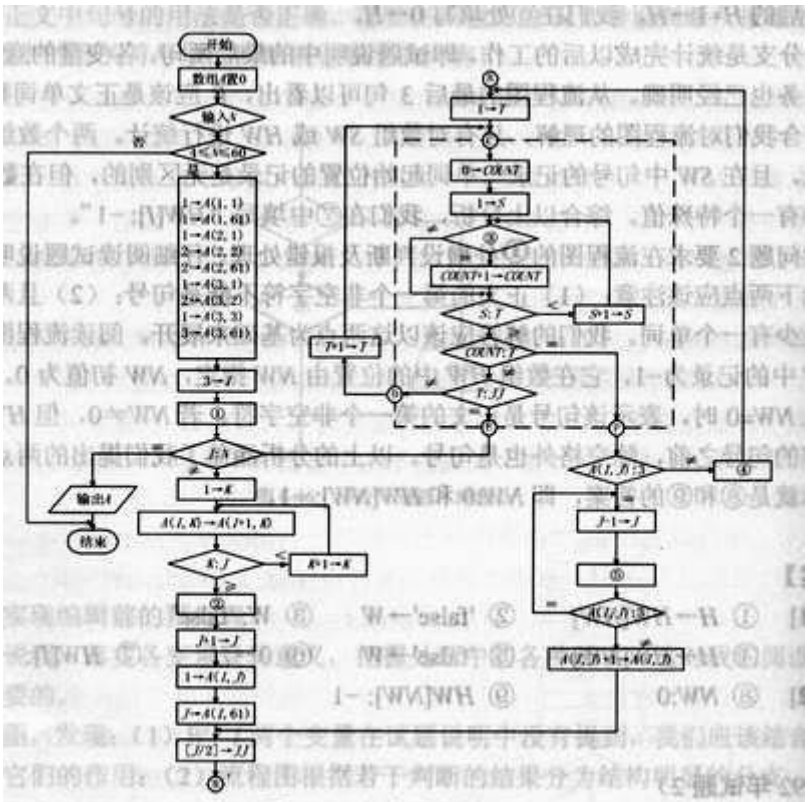


图 6-47 1992 年试题 2 流程图

[说明]

流程图用来按递增顺序产生由数字 1、2、3 组成的非空序列。规定其产生的每个序列为均不存在两个相同的相邻子序列。例如，该程流程图产生的前 5 个序列分：

1
1 2
1 2 1
1 2 1 3
1 2 1 3 1

序列 1211，1212 等均因含有两个相同的相邻子序列，故未被流程图产生。

产生一个序列的方法是，在前一个合理的序列尾部之后添加数字 1，构成一个新序列，然后进行合理性检验。当新的序列不合理时，右新列尾部为 1，则改为 2，若为 2，则改为 3，然后继续进行检验以寻找下一个合理的序列。当尾部已为 3 时，删去该数字后，继续进行上述更改和检验。

本流程图寻找长度大小等 N，（4≤N≤60）的合理序列。二维数组 A 的每一行用于存数找到的一个序列，其中数组元素 A 存放第 1 个序列的长度。

图中[W]表示不超过 W 的最大整数。

［问题 1］

填充流程图中的①～⑤ ，使之成为完整的流程图。

［问题 2］

现对流程图中虚线所围的部分给出了图 6-48 所示的简化形式，试填充其中的⑥～⑧ ，使之成为正确的简化。

【解析】

本题采用回溯技术，按递增顺序产由数字 1、2、3 组成的非空序列，规定其产生的每个序列均不存在两个相同的相邻子序列。

解答问题不一定要拘泥成法，一个简单的逻辑就是能够快速准确地解答问题才是好的解答方法。在阅读试题说明和流程图后，研究从判断 J：N 到 J-A（I，61）之间的流程图，可以推知在 1 和 2 之间的流程图是从上一个序列生产新序列首部的操作。从判断 K：J 及其两个分支可以推知 J 的含义是一个序列的长度（新序列首部的长度），而此前又没有对 J 赋值操作，在程序中不赋值而使用变量是不符合常规的。结合 J 含义，可知 J 的初始值应该为 3，所以的解答应该是给 J 赋初值 3 的任何语句。利用以上对流程图的理解 K≥J 表示上一人序列已经完全复制为新序列的首部，把直到 JA（I，61）之间的内容与试题说明中的产生一个新序列的方法是……添加数字 1″结合起来理解，不难发现这部分实现在上一个序列后添加数字 1 的操作。结合上下文，该处应该是 I 的递加，否则 J-AN（I，J）将会把新序列中现存的最后一个元素（即上一个序列的最后一元素）覆盖掉。

考查流程图基余的部分，不难发现这是进行合理性检验和回溯处理的。从 到判断 A（I，J）：处，是对新序列进行合理性检验的处理，此判断的 A（I，J）=3 分支到 A（I，J）+1A（i，j）是回溯部分。结合试题说明，当新序列不合时……则改为 3，答案显然是 A（I，J）+1A（I，J）；当尾部已 3 时，则删去该数字后，继续进行上述更改和检验，联系语句 J-1-J 序列的长度减 1，对应的 A（I，61）也该减 1，所以凡是实现 A（I，61）所以凡是实现 A（I，61）减 1 的解答都是合理解答。

现在我们集中精力解答，由以上分析可知，所处的部分进行合理性检验，检验的实质是检测该序列中最否存在两个相同的相邻子序列。由于该序列除最后一个元素以外已经是一个合理的序列，所以，合理性检验也就简化为含有最后一位数字的子序列是否有相同的邻子序列。考虑 意义，如果序列中包含有最后一位数字的不序列是否有相同的相邻子序列。考虑 意义，如果序列中包含相邻且相同的两个字序列，那么其最大长度应该是 JJ，因此可以判断 T 为当前检测的子序列的长度。若 T==JJ 且未发现不合理的子序列，则表示检测完毕，该序列通过合理性检验。逐层深入，可以推知 S 为当前检测的子序列中正在检测的位置。根据以上分析，要判断两个序列相同，就要使对应位置的元素全部相同，由此我们又可以推断 COUN 表示位置元素相同的个数。若 COUNT 与当前检测子序列的长度相同，则证明该序列不合理。应该是判断对应位置的元素是否相同，因此我们填写。

COUNT 是必须的吗？只要有一对对应位置的元素不同，即表示当前检测的子序列合理，把试题给出的简化流程图与删除有关 COUNT 语句框后虚框中的流程图进行比较，不准推知的作用是相同的，判断 极为相似，不妨在两处填写 ST 与 S+1-S。然后我们回过头来推敲上述 3 个答案，上述解答完全能够实现子序列的合理性检验，由此我们回过头来推敲上述 3 个答案，上述解答完全能够实现子序列的合理性检验，由此我们印证解答的正确性。

【答案】

［问题 1］①A(3,16)→J ②I+1 →I ③ A(I,J-T-S+1):A(I,J-S+1)

④A(I,J)+1→A(I,J) ⑤J →A(1,61)

［问题 2］⑥A(I,J-T-S+1):A(I,J-S+1) ⑦S:T ⑧为 S+1→S

试题 3 （1994 年试题 3）

阅读下列说明和流程图，如图 6-49 所示，回答问题 1 和问题 2，把解答填入答卷的对应栏内。

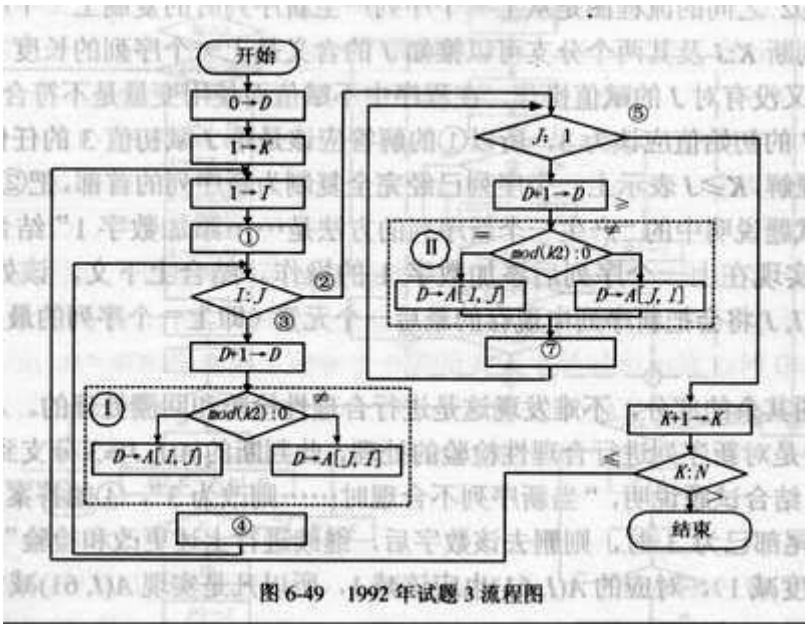


图 6-49 1992 年试题 3 流程图

［说明］

本流程图用来将自然数据按图 6-50 所示的次序依次存放到 N*N 的才维数组 A 中，图中描述了 N=5 时，A 中各元素的值及其赋值次序。

流程图中省略了数据输入和输出。

［问题 1］填充流程图中使之成为完整的流程图。

[问题 2]若将流程图中的两个虚线框部分都改成图 6-51 的处理框，试写出 N=3 时该流程图所产生的数组 A 有各元素值。



图 6-50 1992 年试题 3 说明

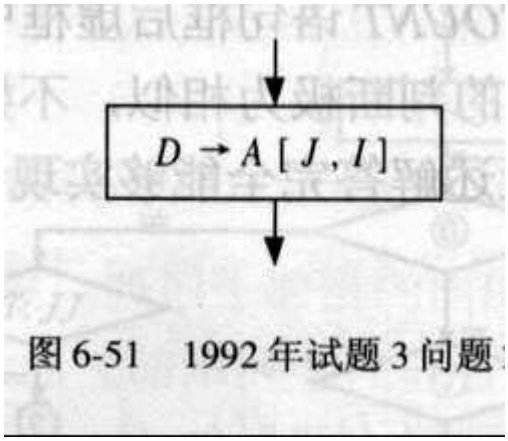


图 6-51 1992 年试题 3 问题

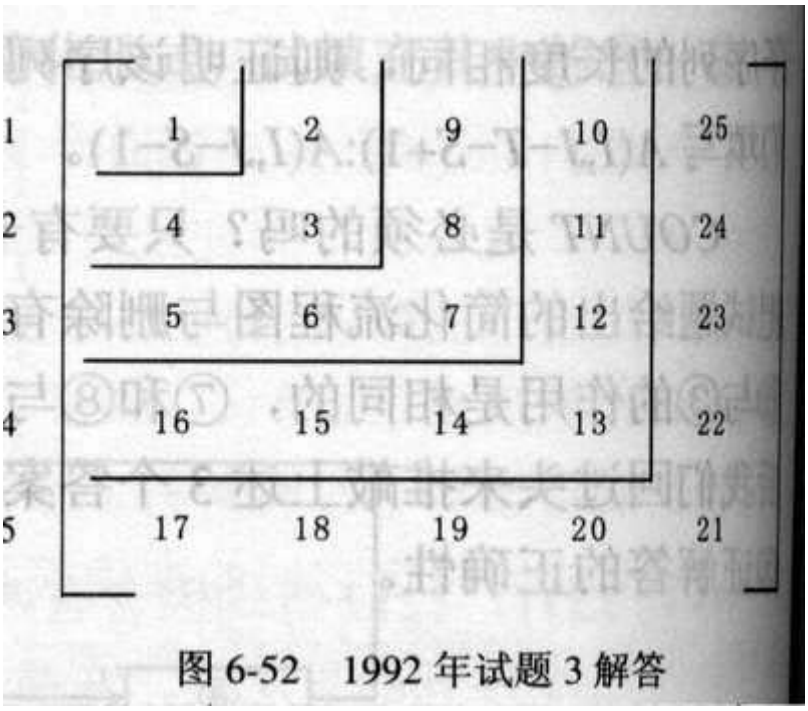


图 6-52 1992 年试题 3 解答

本题是按某特定的次序生成一个 N*N 的二维数组。图 6-52 描述了 N=5 时 5 人区域的划分情况。我们称一个区域为一个直角行。对于 N*N 的二维数组，共有 N 个直角行，其行号如图 6-52 所示，根据题中规定的赋值次序可知，偶数直角行的赋值次序是先赋垂直方向的直角边，后赋水平方向的直角边，我们称它为先列后行。奇数直角行的赋值次序是先赋水平方向的直角边，后赋垂直方向的直角边，我们称它为先行后到的。表 6-9 给出了第 K 直角行的先列后行和先行列的赋值次序。

表 6-9 1992 年试题 3

赋值顺序	1	2	…	k	$k+1$	$k+2$	…	$2k-1$
先列后行	$A[1,k]$	$A[2,k]$	…	$A[k,k]$	$A[k,k-1]$	$A[k,k-2]$	…	$A[k,1]$
先行后列	$A[k,1]$	$A[k,2]$	…	$A[k,k]$	$A[k-1,k]$	$A[k-2,k]$	…	$A[1,k]$

仔细研究试题说明所提供的图，不难发现流程图有 4 点应该注意：（1）处理至第 1 行时，列号加 1，行号为 1；（2）处理至第 1 列时，列号为 1：（3）行号等于列号时，视当前处理情况改变处理方向。（4）在行号与列号码奇数和偶数时处理方向是不同的。下面我们结合流程图理解以上 4 点是如何实现的。

从判断 I：J 和 JI 作为数组 A 的下标来看，实现了第 4 点，由 I：J 和 I：J 引导的两个循环实现了第 3 点，两个循环配合实两第 1-2 点。

现在开始阅读流程图。结合的上下文，此前同有对变量 J 的操作，此后使用了 J，所以应该是对 J 的赋值操作，而且该初值应为 1。I 实现了行号列号为 1 的操作，但行号列号的加 1 呢？显然，如果不借助其他变量的支持，此处是不同时承担赋初值和加 1 两项任务的。考查 K 及流程图的末尾两句，不难推知两点：（1）K 在流程图的大循环中递加；（2）K 与行数列数（N）关系密切，由 K 与 N 的比较

来判断程序是否结束。结合 K 的初值为 1，我们在处填写 K-J，实现赋初值和加 1 两项任务。

当我们以 1 和 2 地进入系统时，发现两点：（1）如果 1、2、3 与 5、6 的分支导出情况一致，那么无论是 1-A 还是，I 总是不能进入数组 A，所以我们判断两个判断的分支导出情况应该正好相反。的判断结果不应该是″=″和″≠″，否则只有行号等到于列号的数组元素才能被操作。但 1 的操人作应该在第 1 个循环中实现还是在第 2 个循环中实现呢？很明显，如果在第 1 循环中实现，则在完成 1 后修改列号（由试题说明可知），不可避免地又会进入第 2 个循环，因此我们只好反 1 放在第 2 人循环中实现。这样，我们在 5 和 6 中填写或来实现。或放在 6 处是为了保证 1 的实现。J 在 7 中是无法通过合理的修改达到 J 的，详细原因不在此 述。所以我们选择，这样 2 和 3 的解答也就出来了。

现在剩下 4 和 7 没有解答，显然，这里是在修改行号或列号，答案无非是 I 或 J 的加 I 或减 1，但如何确定他们的答案？可以考虑，如果 4 不是 I 的递加，则循环就会陷入死循环；同样的思路我们可以判定的解答是 J 的递减。

题中的问题 2 把图中的两个虚线框改成 D-AJ，根据我们对流程图的理解可知，D 是先行后列的赋值次序，这样修改后的流程图总是按先行后列的次序进行赋值。因此，当 N=3 时流程图所产生的数组 A 的各元素值为 149、238、567。

【答案】

［问题 1］①K→J ②≥ ③< ④I+1→I ⑤< ⑥≥ ⑦J-1→J

［问题 2］149 238 567

试题 4 （1992 年试题 4）

阅读下列说明和流程图 6-53，回答问题 1 至问题 3。

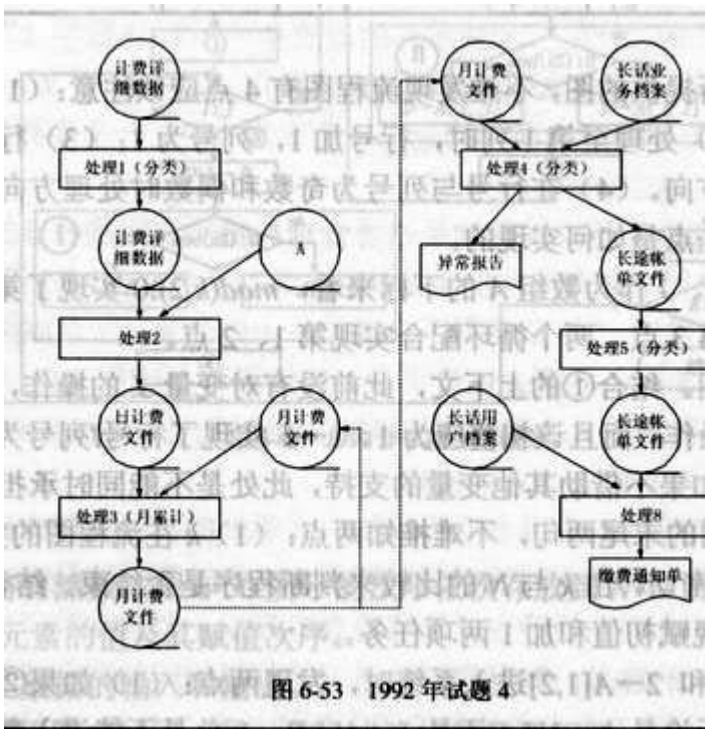


图 6-53 1992 年试题 4

[说明]

（1）流程图描述了某电话局长途电话业务及计费管理系统中的计费数据处理流程。

（2）每个用户在系统的"长话用户档案"中有一个记录，该记录由用户编码唯一标识。一个用户可能拥有多个可使用长途直拨业务的电话号码（称为有权电话号码）。每个有权电话号码在"长话业务档案"中有一个记录。它们分别含有以下数据项。

"长话用户档案"：用户编码、用户名称、用户地址

"长话业务档案"：电话号码、用户编码、国内长途许可标志、国际长途许可标志

（3）计费详细数据由电话号码程控交换机自动记录在磁带上，用作系统的输入。系统每天晚上零时处理计费详细数据磁带，计算通话次数和电话费，产生日计费文件，并把这些数据累计到"月计费文件中"。

"计费详细数据"包括以下数据项：

电话号码、受电话号码、日期、通话开始时间、通话持续时间

系统根据受电话号码可以区分国内长途和国际长途。

（4）系统每月 1 日上午进行出帐处理，给每个月户提交一份上月的包括以下内容的"缴费通知单"。每个有权电话号码每月还需向电话局付"门号费"10 元。

×××电话局长途电话用户缴费通知单

用户编码： 用户名称： 用户地址：

电话号码	国内通话次数	国内金额	国际通话次数	国际金额	门号费	金额合计
…	…	…	…	…	…	…

日期 ××年××月 金额总计

[问题 1]

除了上述说明中指明的文件外，流程图中还需用到文件 A。指出文件 A 应是什么文件？

[问题 2]，

指出“日计费文件”至少应包括哪些数据项？

[问题 3]

①指出处理 5 分类的第一、第二关键项。

②指出“长话业务档案文件”应按哪一个关键项分类？

【解析】

本题描述了长途电话计费的处理流程。

仔细阅读试题说明和流程图，不难发现，处理 2 使用计费详细数据和文件 A 生成日计费文件，日计费文件与月计费文件合并后形成每月的月计费文件。很明显，计费详细数据中只提供了计费的一人参加数，即通话持续时间，根据常识，应该有一个话费单价作为计费参加数。结合流程图的后半部分，可以推知在形成月计费文件之前不需要其他数据文件的参加，所以判定文件 A 是电话费单价文件。

考查日计费文件的数据数源和数据输出流向，发现日计费文件的数据来源是计费详细数据和话费单价文件，日计费文件中的数据应该是可以直接或间接从计费详细数据取得的，日计费文件最后合并入月计费文件，可知两者的文件结构是大体相同的。由下面的月计费文件与长话业务档案相配合生成的缴费通知单的结构可知，月计费文件至少应该包括电话号码、国内通话次数、国内金额、国际通话次数和国际金额，我们就把它当作日计费文件的结构。考查这些数据项，它们都可以直接或间接从计费详细数据取得，应该是合理的。

处理 5 是对长话帐单文件进行分类，分类后要与长话用户档案配合通过处理 8 产生缴费通知单，长话用户档案应该是按用户编码有序的，长话帐单文件是由月

计费文件和长话业务档案相配合而产生的。从缴费通知单可以推断两点：（1）缴费通知单按用户编码产生，每张通知单上有若干个电话号码；（2）长话帐单文件中应该而且能够包含电话号码和用户编码。为使处理 8 能够获得理想的处理速度和处理效率，用户编码应该是长话帐单文件的第一关键项，电话号码应该是长话帐单文件的第二关键项。

长话业务档案可以电话号码或用户编码为关键项进行分类，但考虑到长话业务档案与月计费文件配合产生长话帐单文件，即使不考虑处理 4 的内容，当两个文件配合参与处理时，使用相同的关键项分类能够使处理获香港理想的处理速度和处一效率，从上面的分析我们可以推知，月计费文件只能以电话号码为关键项进行分类，所以长话业务档案必须电话号码为关键项进行分类。

【答案】

［问题 1］电话费单价文件

［问题 2］电话号码、国内通话次数、国内金额、国际通话次数、国际金额

［问题 3］①第一关键项是用户编码，第二关键项是电话

②电话号码

试题 5（1992 年试题 5）

阅读下列说明和流程图，如图 6－54 所示，回答问题，把解答填入答卷的对应栏内。

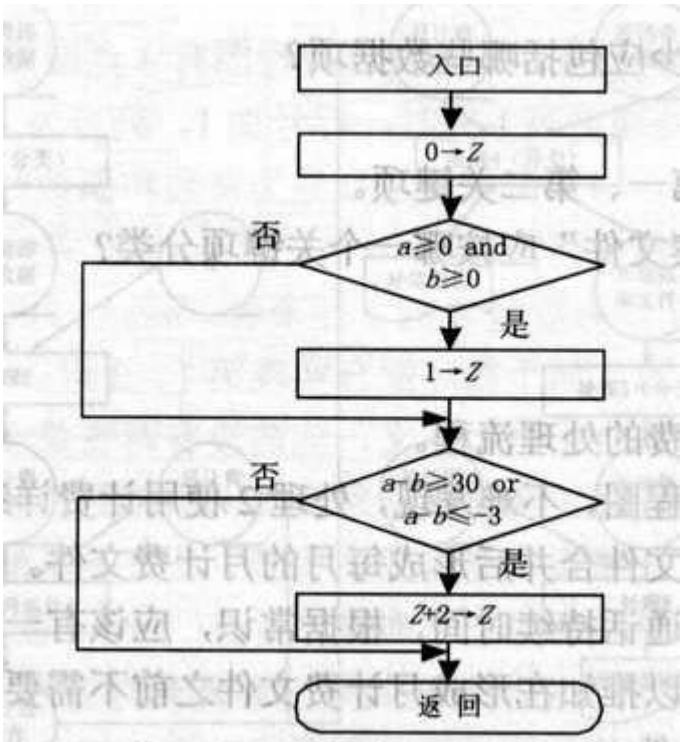


图 6-54 1992 年试题 5 流程图

［说明］

本流程图描述了某子程序的处理流程，现要求用白盒测试法对子程序进行测试。

［问题］

根据判定覆盖、条件覆盖、判定 / 条件覆盖、多重条件覆盖（条件组合覆盖）和路径覆盖 5 种覆盖标准，从供选择的答案中分别找出满足相应覆盖标准的最小的测试数据组（用①～⑩表示）。

供选择的答案

①a=5 b=1

②a=5
b=-1

③a=5 b=1
a=-5 b=-1

④a=5 b=1
a=0
b=-1

⑤a=5 b=-1
a=-5 b=1
a=-5 b=-1

⑥a=5 b=1
a=0 b=0
a=-5
b=-1

⑦a=5 b=1
a=0 b=1
a=0 b=-1

⑧a=5 b=1
a=0
b=-1
a=-5

	b=1
a=-5 b=1	a=-5
	b=-1
⑨ a=5 b=1	⑩a=5 b=1
a=0 b=-1	a=5 b=0
a=0 b=1	a=5
	b=-1
a=-5 b=1	a=0 b=1
a=-5 b=-1	a=0 b=0
	a=0
	b=-1
	a=-5
	b=1
	a=-5
	b=0
	a=-5
	b=-1

【解析】

本题的关键是为该流程图的白盒法测试设计数据，解答本题的基础是对判定覆盖等 5 种覆盖标准的把握。

首先我们列出 5 种覆盖标准的要求。判定覆盖标准是指提供足够的测试用例使和每个判定的每一种可能的结果至少出现一次。条件覆盖标准是指提供足够的测试用例使得每个条件的所有可能的结果至少出现一次，并且每个判定本身的所有可能结果至少出现一次，也就是说，判定条件覆盖是既满足判定覆盖标准，又满足条件覆盖标准。多重条件覆盖标准是指提供足够的测试用，使得每个判定中条伯结果的所有可能组合至少出现一次。

分析流程图，列出流程图中所有的判定和条件，建立真值表（见表 6-10），把各组数据在各个判定或条件上的真或假在表中列出，试题的解答就非常明显了。这也是解答此类题目的一种单方法。我们以 T 表示该组数据在该判定或条件上为真值，F 表示该组数据在该判定或条件上为非真值。

表 6-10 1992 年试题 5 真值表

测试数据	判定		条件			
	a≥0 and b≥0	a-b≥3 or a-b≤-3	a≥0	b≥0	a-b≥3	a-b≤-3
①	T	T	T	T	T	F
②	F	T	T	F	T	F
③	T/F	T/T	T/F	T/F	T/F	F/T
④	T/F	T/F	T/T	T/F	T/F	F/F
⑤	F/F/F	T/T/T	T/F/F	F/T/F	T/F/F	F/T/T
⑥	T/T/F	T/F/T	T/T/F	T/T/F	T/F/F	F/F/T
⑦	T/T/F/F	T/F/F/T	T/T/T/F	T/T/F/T	T/F/F/F	F/F/F/T
⑧	T/F/F/F	T/F/T/T	T/T/F/F	T/F/T/F	T/F/F/F	F/F/T/T
⑨	T/F/T/F/F	T/F/F/T/T	T/T/T/F/F	T/F/T/T/F	T/F/F/F/F	F/F/F/F/T
⑩	T/T/F/T/T/F/F/F/F	T/T/T/F/F/F/T/T/T	T/T/T/T/T/T/F/F/F	T/T/F/T/T/F/T/T/F	T/T/T/F/F/F/F/F/F	F/F/F/F/F/F/T/T/T

结合以上关于覆盖标准的叙述和真值表，我们可以解答满足判定覆盖、条件覆盖、判定/条件覆盖和多重条件覆盖 4 种标准要求的测试数据。

路径覆盖标准是指提供足够的测试用例，使每条路径都至少执行一次。本题中共有 4 条路径，因此至少应有 4 个测试数据。在 8 和 7 之间比较，不难分析出测试数据 7 满足路径覆盖标准。

【答案】

判定覆盖 ④

条件覆盖 ③

判定条件覆盖 ⑥

多重条件覆盖 ⑧

路径覆盖 ⑦

6.10 1991 年度软件设计试题解析

试题 1 是对一个递归函数的实现，试题 2 是关于相乘的问题，相对来说，试题 1 的难度要低一些，至少要实现的算法是明确的。试题 1 的难度要低一些，至少要实现的算法是明确的。试题 2 则不同，解答题目首先必须明确流程图所采用的算法，然后在此基础上解答试题，这对考生的能力要求较高。

试题 3 是关于分层数据流图的分析和修改。这类试题只要贯彻数据平衡原则，把握各层流程图之间的对应关系，解答难度就会相对低些。

本年度软件设计试题较有特色。试题 4 的问题 3 设计巧妙，要求考生在掌握流程图的基础上彻底修改流程图，以实现另外的功能；试题 5 的问题 2 也很特殊，要求考生修改文件的数据项以解决一个在流程图中没有解决的问题。

试题 1 （1991 年试题 1）

阅读下列说明和流程图 6-55，回答问题 1 和问题 2，把解答填入答卷的对应栏内。

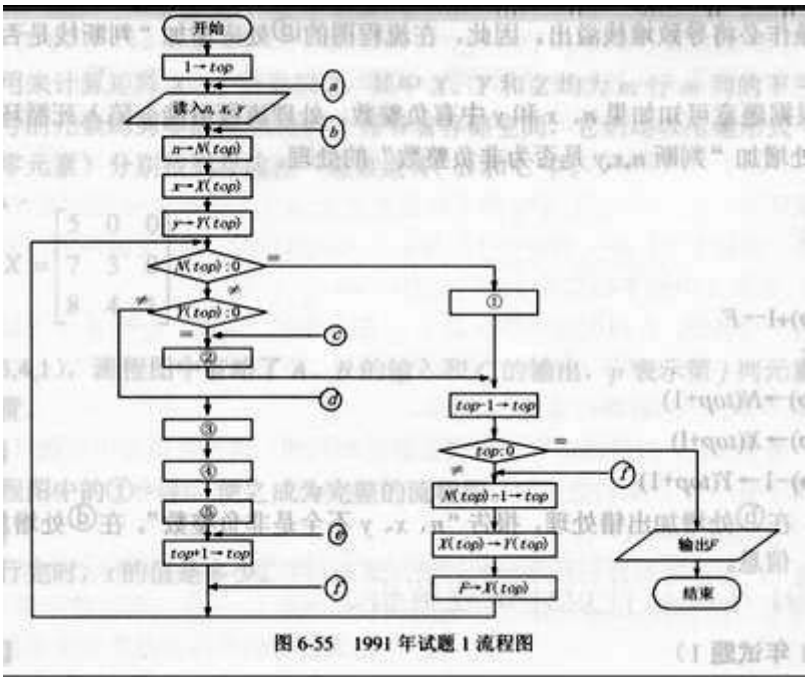


图 6-55 1991 年试题 1 流程图

［说明］

流程图实现下列递归函数的计算。递归函数定义如下：

A(n, x, y)=

x+1	当 n=0
1	当 n≠0 且 y=0 时
A(n 1, , A(n, x, y-1), x)	当 n≠0 且 y=0 时

其中 n, x, 和 y 均为非负整数。

递归函数的计算使用了 3 个栈，它们分别用数组来存放，为栈顶指针，中间计算结果和最终计算结果均存放在变量 F 中。

［问题 1］

填充流程图中的① ～⑤，使之成为完整的流程图

［问题 2］

指出应在流程图的哪些位置（用 a～g 表示）上增加检测错误的流程图，并分别指出这些位置能报告哪些错误信息。

【解析】

该题实现递归函数的计算，计算中使用了堆栈。堆栈 N、X 和 Y 分别用来存放函数的自变量 n、x、和 y。栈顶元素 N(top)、X(top)和 Y(top)存放的是当前正在计算的函数的自变量。

本题的流程图是对递归函数的实现，我们在解答时要紧密结合递归函数的算法，按照递归函数的的算法来确定各个需要解答的问题，所以解答之前要仔细研究递归函数的算法，为正确理解流程图打下基础。

考查流程图的结构，结合递归函数的定义，可知当 n=0 时函数值为 x+1，当 n≠0，且 y=0 时函数值为 1。流程图中的 1 和 2 应该是对此的实现。所以 1 和 2 的答案分别为 X(top)+1-F 和 1-F。

当 n≠0 且 y=0 时，函数值为 A(n-1,A(n,x,y-1),x)的值，此时应先计算 A(n,x,y-1)的值 F，然后计算 A(n-1,F,x)即当 n≠0 且 y≠0 时需调用递归函烽 A 两次。第一次调用时，自变量 n、x 和 y-1 需要时栈。当计算出 A （n,x,y-1）的值后。栈顶元素需退栈。第二次调用 A 时，可把此时的栈顶元改为 F 和 x。当 A（n-1,F,x）计算完再时把栈顶的自变量退栈。根据以上分析，流程图的 3、4、5 应分别为 N(top) N(top)、X(top)、X(top+1)和 Y（top）-1Y(top+1)。

在堆栈操作中，比较常用的操作就是在进栈之前判断堆栈是否已满。在堆栈已满的情况下进行栈操作必将导致栈溢出。因此，在流程图的 d 处应增加"判断栈是否已满"的处理。

此外，根据题意可知如果 n、 x 和 y 中有负整数，处理流程可能会陷入死循环，因此应在流程的 b 处增加判断 n，x，y 是否为非负整数的处理。

[答案]

［问题 1］

①X(top)+1→F

②1→F

③N(top)→N(top+1)

④X(top)→X(top)+1

⑤Y(top)-1→Y(top+1)

［问题 2］在(b)处增加出错处理，报告“n,x,y 不全是非负整数”。在(d)处增加出错处理，
报告“栈满”信息。

试题 2 （1991 年试题 1）

阅读下列说明和流程图，如图 6－56 所示，回答问题 1 和问题 2，把解答填入答卷的对应栏内。

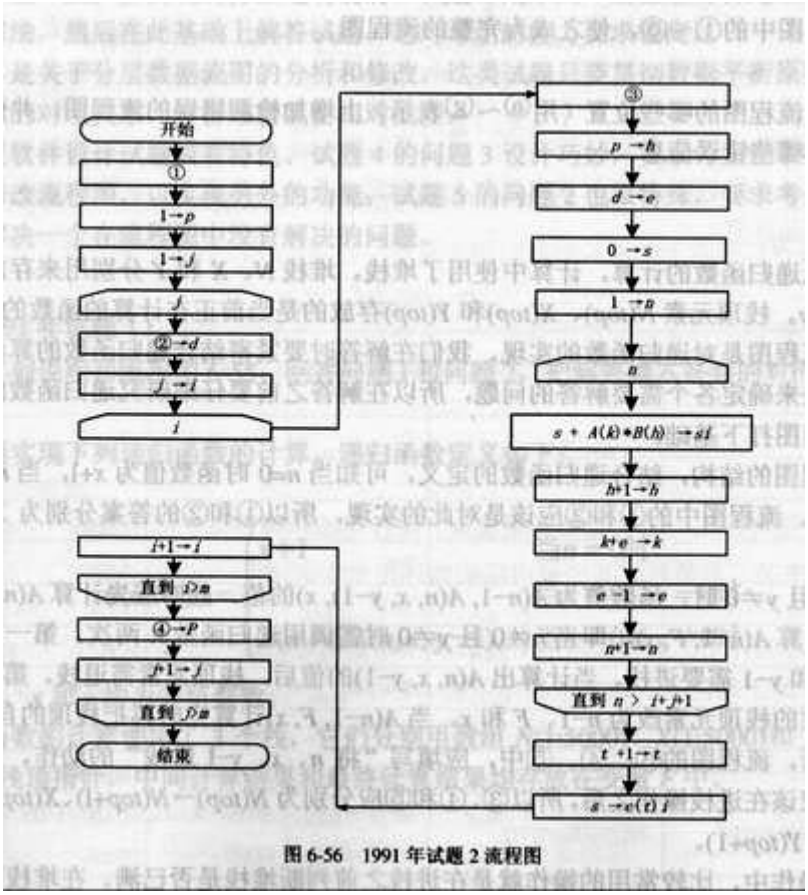


图 6-56 1991 年试题 2 流程图

［说明］

流程图用来计算矩阵 X、Y 的乘积 Z，其中 X、Y 的均为 m 行 m 列的下三角方阵。即行号小于列号的元素均为零的 m 阶方阵。为节省存储空间，它们均以压缩形式（不存放矩阵上三角中的零元素）分别按列存放在一维数组 A、B 和 C 中。

例：若 $X=\begin{bmatrix}5 & 0 & 0\\ 7 & 3 & 0\\ 8 & 4 & 1\end{bmatrix}$

则 A=（5，7，8，3，4，1）流程图中省略了 A、B 的输入和 C 的输出，p 表示第 j 列元素在一维数组中的起始位置。

［问题 1］

填充流程图中的 1、4，便之成为完整的流程图。

［问题 2］

程序执行完时，t 的值是多少。

[解析]

解答本题需要一定的数学知识，并结合流程图的结构，考查注流程图中矩阵相乘算法的实现，两方面要融会贯通，否则很难解答此类题目。

本题是计算两个下三角方阵 X、Y 的乘积 Z，其中 X、Y 和 Z 都有以压缩形式按列存储在数组 A、B 和 C 中。首先，求未经压缩的两个下三角方阵 X、Y 的乘积 Z，其计算公式为：

$$Z_{ij}=\sum_{r=1}^{m} Xir * Yrj \quad (i,j=1,2,\cdots,m) \quad (1)$$

由于 Z 也是一个下三角方阵，在压缩存储的情况下只需计算 Z 的下三角部分。因此，公式（1）可变化为：

$$Z_{ij}=\sum_{r=1}^{m} Xir * Yrj \quad (j=1,2,\cdots,m;\ i=j,j+1,\cdots,m) \quad (2)$$

由于 X 和 Y 均为下三角方阵，即当 i< r 时，Xir=0; 当 i<j 时，Yrj=0 ,所以公式（2）又可变化成：

$$Z_{ij}=\sum_{r=1}^{m} Xir * Yrj \quad (j=1,2,\cdots,m;\ i=j,j+1,\cdots,m) \quad (3)$$

公式（3）现可以用一个三重循环来实现。结合对流程图整体结构的理解，它们应该分别对应于流程图中的三重循环，其中最内层的循环用来实现 $\sum_{r=1}^{m} Xir * Yrj$ 的计算。

明确了循环的作用，解答本题的关键就集中在如何根据循环变理找出 Xir 和 Yir 在下三角方阵 A 和 B 的正确位置。由中层循环中的 t+1→t 和 s→c(t)可以推知，

公式（3）的计算顺序应该是首先计算第一列的元素，再计算第二列的元素…据此，t 的初值应为 0。

显然，下三角方阵 X 或 Y 中第 1 行第 j 列的元素在 A 或者 B 中的下标应该是下面的数据列：m,m-1,…m-j+1。这样，下三角方阵 X 或 Y 中第 I 行第 j 列的元素在 A 或者 B 的下标应该是 m+(m-1)+…(m-j+2)+i-j+1。简写这个算式，应该是 j(2m-j+1)/2+i-m.

结合对流程图结构和计算方法的分析，可以推知，在最外层循环中，当 j =1 时，公式（3）中 Yrj 为 y11，它在 B 中的下标为 1。由上述的下标计算公式可以得出如一结论：j 每增加 1，在 B 中的下标值应增加 m-j+1，流程图中的④→p 应该是对下标的修改，因此④应为 p+m-j+1.在中层循环中，Yr,j 是从 Yjj 开始,与 p→h 相对应。在内层循环中，$Y^{r+1,i}$ 与 Yrj 在 B 的下标值之差为 1，与 h+1→h 相对应。

在内层循环中，Xi,r+1 与 Xir 在 A 中的下标值之差为 m-r,与 k+e→k 相对应，其中 e 即为 m-r 的值。当 n 增加 1 时，m-r 就减少 1，当 e-1→e 相当应，在中层循环中，公式（3）的 xir 从 xij 开始，它在 A 中的下标值为 j(2m-j+1)/2+I-m。

从流程图中可看到，在内层循环中 p 表示 yij 在 B 中的下标值，xjj 在 A 中下标的表示应该是相同的，因此 xjj 在 A 中的下标值也表示为 p+I-j>k。

对于同一个 j 值，内层循环中 e 的初值都是相同的，这也可以由中层循环中的 d→e 推知。在外层循环中，为 d 赋初值是我们必须解答的问题，从上面的分析可知，d 中存放的是 m-j，所以我们在②处填入 m-j.

解答问题 2，可以通过分析程序的执行情况来求得答案。在程序执行完时，t 应该是 zmm 在 C 中的下标值，所以 t=m+(m-1)+…+1=m(m+1)/2。

[答案]

[问题 1]

①0→t ②m-j

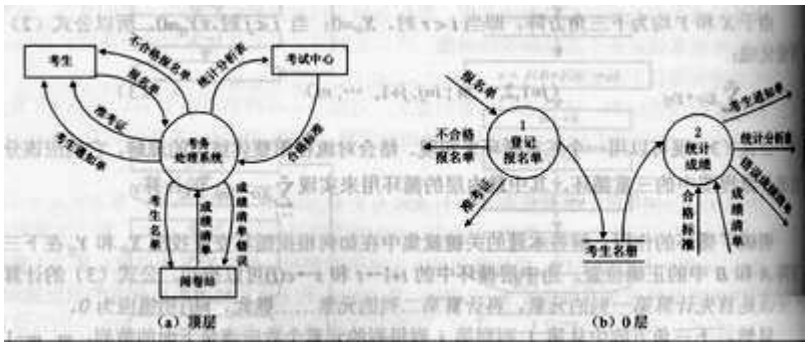
③p+I-h→k 或 j*(2m-j+1)/2+I-m→k 或 (j-1)(2n-h+2)/2+I-j+1→k

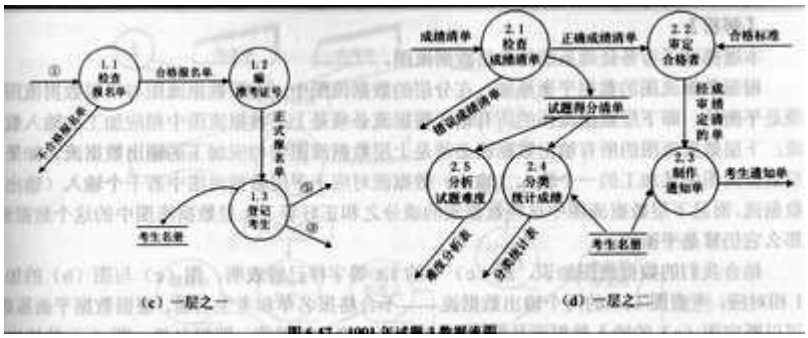
④p+m-j+1 或 j*(2m-j+1)/2+1

[问题 2] m(m+1)/2

试题 3 （1991 年试题 3）

阅读下列说明和流程图，如图 6-37 所示，回答问题 1 至问题 4，把解答填入答卷的对应栏内。





[说明]

流程图是采用结构化分析方法画出的某考务处系统的数据流程图（DFD），图中圆圈表示加工：→表示数据流；□表示数据源终点；═表示文件。该系统有如下功能；

（1）对考生送来报名单进行检查。

（2）对合格的报名单编好准考证号后将准考证送给考生，并将汇总后的考生名单送给阅卷站。

（3）对阅卷站送来的成绩清单进行检查，并根据考试中心制订的合格标准审定合格者。

（4）制作考生通知单送给考生。

（5）进行成绩分类统计（按地区、年龄、文化程度、职业和考试级别等分类）和试题难度分析，产生统计分析表。

部分数据流的组成如下所示：

报名单=地区+序号+姓名+性别+年龄+文化程序+职业+考试级别+通信地址

正式报名单报名单+准考证号

准考证=地区+序号+姓名+准考证号+考试级别

考生名单=（准考证号+考试级别）（其中{w}表示 w 重复多次）

统计分析表=分类统计表+难度分析表

考生通知单=考试级别+准考证号+姓名+合格标志+通信地址

[问题 1]

指出一层之一（图(c)）的数据流图中①、②、③的数据流名。

[问题 2]

指出 0 层（图(b)）的数据流图中有什么成分可删去。

[问题 3]

指出一层之二（图(d)）的数据流图中在哪些位置遗漏哪些数据流，也就是说，要求给出漏掉了哪个加工的输入或输出数据流的名字。例如，加工 2。5 的输出数据流"难度分析表"。

[问题 4]

指出考生名册文件的记录至少包括哪些内容。

[解析]

本题描述了考务处理系统的分层数据流图。

根据数据流图的数据平衡原则，在分层的数据流力部，上层数据流图与下层数据流图必须是平衡的，即下层数据流图的所有输出数据流必须是上层数据流图中相庆加工厂的输出数据流。如果上层数据流力部某加工的一个输入（输出）数据流对庆于下层数据流图中若干个输入（输出）数据流，而且下层数据流图中这些数据流的成分之和正好等于上层数据流力部的这个数据流，那么它仍算是平衡的。

结合我们数据流图知识，图（c）中的 1。X 等字样已经表明，图（c）与图（b）的加工 1 相对应，考查图(c)的两个输出数据流--不合格报名单和考生名册，根据数据平衡原则，可以断定图（c）的输入数据流是图（b）加工 1 的输入数据流，即报名单。图（c）的输出数据流应该与图(b)加工 1 的输出数据流等价，所以图(c)的输出数据流②和输出数据流③应该是准考证和考生名单。

上面对图（b）加工 1 已经进行细致的分析，未发现可删除的成分。考查图（b）加工 2 的输入数据流和输出数据流，发现试题得分清单并不是系统功能所要求的，虽然在图（d）加工 2。1 之后用来产生难度分析表和分类统计表，但只是在

加工时使用试题得分清单，完全可以从加工 2。1 之后用来产生难度分析表和分类统计表。由此我们断定图（ b）的输出数据流试题得分清单是可以删除的。

显然，图（ d）是与图（b）的加工 2 相对应的。根据数据平衡原则，考查图(b)加工 2 的输入数据汉和输出数据流，发现图(d)中缺少输入数据流合格标准和输出数据流错误成绩单。仔细考查图（d），易知输出数据流错误成绩清单应该从加工 2。1 流出，而输入数据流合格标准应该流入加工 2。。

仔细阅读数据流力图，可知考生名册文件的数据源是正式报名单，并在加工 2。3 中产生考生通知单，在加工 2。4 中产生分类统计表。这样，考生名册文件数据项的来源和应用范围都已确定。结合试题说明，首先将考生通知单中除合格标志外的数据项都包括时考生名册文件。成绩要按地区、年龄、文化程度、职业和考试级别分类统计，这些数据项都在（正式）报名单中，而加 2。4 又没有使用（正式）报史单，显然，以上 5 个数据英也要包括进考生名册文件。

[答案]

[问题 1]①报名单元 ②准考证 ③考生名单

[问题 2]文件试题"得分清单"可删除

[问题 3] 加工 2.1 遗漏输出数据流"错误成绩清单"，加工 2。2 遗漏输入数据流"合格标准"

[问题 4]

考生名册=地区+姓名+年龄+文化程度+职业+考试级别+通信地址+准考证号

试题 4

阅读下列说明和流程图，如图 6-58 所示，回答问题 1 至问题 3，把解答填入答案的对应栏内。

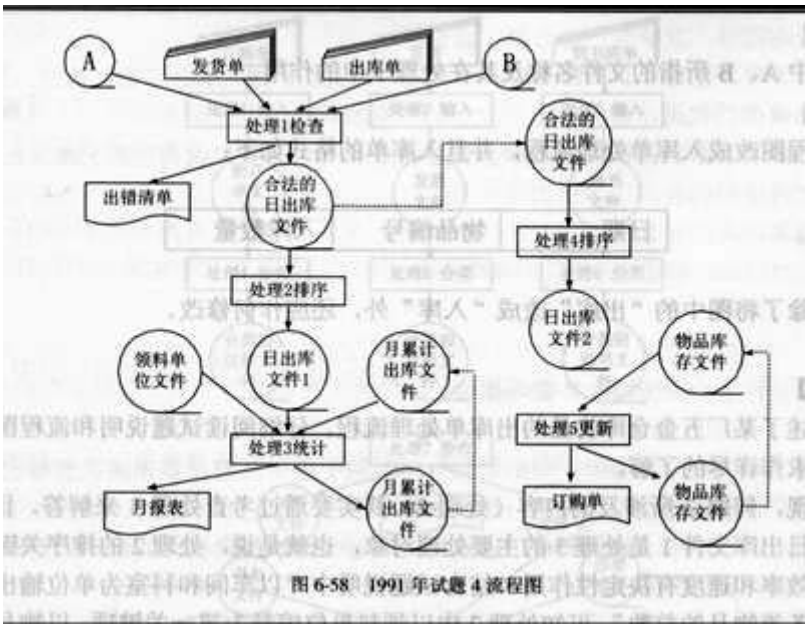


图 6-58 1991 年试题 4 流程图

[说明]

流程图描述了某厂五金仓库物品的出库存单处理流程。该厂有 50 个车间和科室，五金仓库内有近万种物品。仓库管理人员凭入库单收进物品，凭出库单发出物品。系统要求每天根据出库存单更新物品库存文件。当某物品更新后的库存量低于该物品的最低库存量时，输出订购清单。每月还要以车间和科室为单位输出该部门一个月内领取各类物品的总数（称为月报表）

规定每张出库单由两位操作员分别输入，以检查操作员的输入错误。

物品库存文件按物品编号排序，其格式如下：

物品编号	名称	规格	库存量	最低库存量	最高库存量

其中"最高库存量"是指该物品允许存放在库中最大值。

领料单位文件按领料单位编号排序，其格式如下：

领料单位编号	领料单位名称

出库格式如下：

日期	物品编号	出库数量	领料单位编号

[问题 1]

指出处理 2 排序时的第一关键项和第二关键项。

[问题 2]

指出图中 A、B 所指的文件名称及其在处理 1 中作用。

[问题 3]

若把流程图改成入库单处理流程，并且入库单的格式如下：

日期	物品编号	入库数量

那么，除了将图中的"出库"改成"入库"外，还应作任何修改。

[解析]

本题描述了某厂五金仓库物品的出库单处理流程。仔细阅读试题说胆和流程图，对系统的功能和要求作详尽的了解。

不难发生，问题 2 所涉及的内容（处理 2）其实要通过考查处理 3 来解答，因为处理 2 排序产生的日出库文件 1 是处理 3 的主要处理对象，也就是说，处理 2 的排序关键项的设置来处理 3 的效率和速度有决定性作用。结合试题说明中"以车间和科室为单位输出该部门一个月内领取各类物品的总数"，可知处理 2 中以领料单位编号为第一关键项，以物品编号为第二关键项符合处理 3 的要求，能够提高处理 3 的处理速度和处理效率。

处理 1 要对出库单进行检查，为保证数据的正确性，应该对出库存单的每一数据项时行检查，日期的检查是很容易的。但物品编号、出库数量、领料单位编号合法性，必须使用物品库存文件，要难出库存数量、领料单位编号如何检查，显然应该借助文件 A 和文件 B 进行，考查试题说明所给出的文件结构，要验证物品编号的合法性，必须使用物品库存条件，要验证出库数量合法性（如是否大于库存量），必须使用物品库存文件，要验证领料单位编号的合法性，必须使用领料单位文件。综合以上分析，文件 A 和文件 B 所指的对象应该是领料单位文件和物品库存文件。

如果把本流程图改成入库存单处理流程，由于入库过程不中不存在领料行为，所以有关领料和领料单位的操作都应该删除或改成入库操作。处理 3 可以改成月入库统计、报表。另外，处理 5 产生 的订购单显然是不应该在入库过程中出现的。考试到试题说明中"最高库存量"是指该物品允许存放在库中最大值。应该将此处改成超过是高库存量物品报告。

[答案]

[问题 1]第一关键项："领料单位编号"；第二关键项"物品编号"。

[问题 2]A 所指的是领料单位文件，其作用是检查出库单中是否有非法物品编号和非法出库数量（例如出库量>库存量）。

[问题 3]删除图中所有领料单位文件，并把处理 5 输出的订购单改为"报告超过最高库存量信息"。

试题 5 （1991 年试题 5）

阅读下列说明和流程图，如图 6-59 所示，回答问题 1 至问题 3。

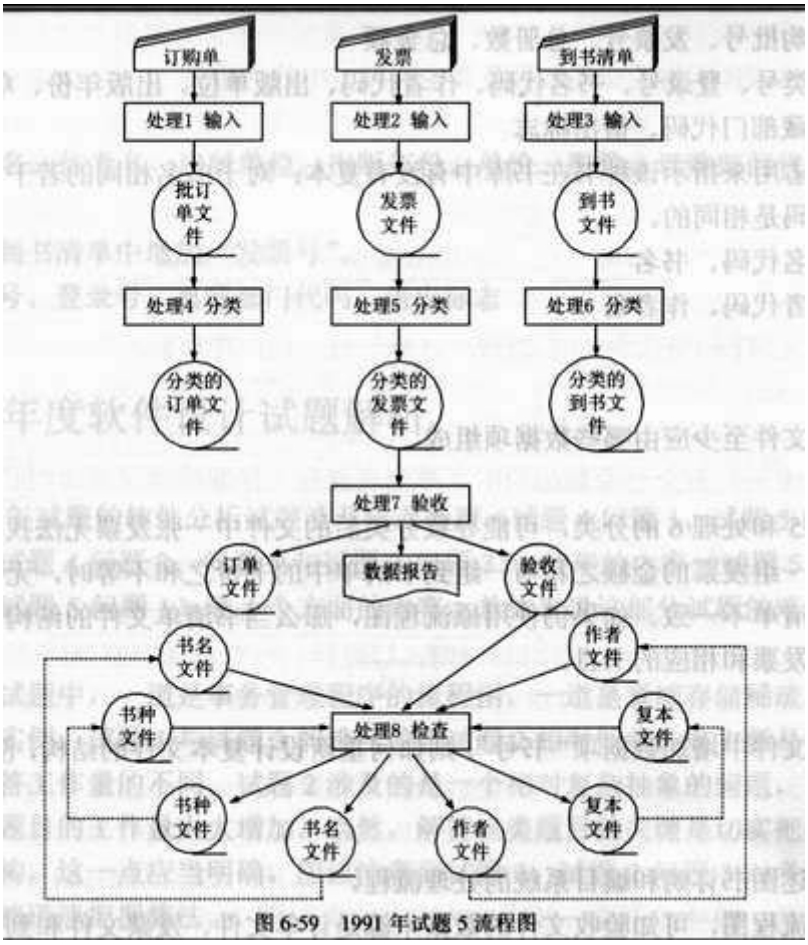


图 6-59 1991 年试题 5 流程图

[说明]

流程图描述了某高校图书订购与编目系统的处理流程。全校的图书典藏在校图书馆和各系的资料室中。学校每年分若干批向出版单位订购图书，同一批订购的图书将陆续邮寄到学校。出版单位在寄出图书的同时附上到书清单和发票，发票上仅给出一份到书清单中书的总册数和总金额。学校收到图书和发票后，先参照订购单验收，然后进行编目，并把有关信息存放在书种文件、书名文件、作者文件和复本文件中，以供读者检索。

书种文件记录了每种书的有关信息。所谓一种书是指同一作者、同一书名、同一出版单位和同一出版年份出版的书。例如，1990 年王某在高等教育出版社出版了《高等数学》（印数 8000 册）和《高等数学》（印数 5000 册），则王某在 1990 年出版了两种书。在全校的藏书中，如果一种书只有一册，则该书的信息存放在书种条件中;如果一种书有多册,则其中一册书的信息存放在书种文件中，

其余的书作为复本将信息存放在复本文件中。复本文件的结构一与书种文件的结构相同。每种书都有一个记号，书号唯一地标识了一种书。在书库中，每册书有一个登陆号，登陆号唯一地标识了一册书。此外，为了图书检索的方便，将图书按学科分类，分类号用来标识不同的学科领域。

各类单据和文件的结构如下：

订购单：订购批号、书名、作者名、出版单位、出版年份、单价、订购册数、订购部门代码、订购日期

到书清单：订购批号、书名、作者名、出版单位、出版年份、单价、册数

发票：订购批号、发票号、总册数、总金额

书种文件：分类号、登录号、书名代码、出版单位、出版年份、单价、复本标志、典藏部门代码、借出标志。

其中，复本标志用来指示该种书在书库中没有复本；对于书名不同若干种书，书名的代码是相同的。

书名文件：书名代码、书名

作者文件：作者代码、作者名

[问题 1]

指出验收文件至少应由哪此数据项组成。

[问题 2]

由于处理 5 和处理 6 的分类,可能导致分类在后的文件中一张发票无法找到与定它对应的那些书,从而当一组发票的金额之和与一组到书清单中的书价之和不等时,无法知道是哪一张发票和哪一份清单不一致。如果仍使用原流程图，那么当书清单文件的结构作何改动后，能找出不一致的发票和相应的书目。

[问题 3]

若在书种文件中增加数据"书号"，则如何重新设计复本文件的结构，使数据冗余最小。

【解析】

本试题描述图书订购和编目系统的处理流程。

仔细研究流程图，可知验收文件的数据来源是订单文件、发票文件和到书文件，因此可以确定验收文件数据项的最大范围上述 3 个文件数据项的并集。考查处理

8，发现除了验收文件外，处理 8 更新了书种文件、书名文件、作者文件和复本文件 4 个文件，因此凡是处理 8 所需而又不包含在上述 4 个文件中的数据项都应该存放在验收文件中。显然，像订购批号、订购日期、总册数和金额等数据项对处理 8 是没有意义的，不应该包含在验收文件中。至于分类号、登录号、书名代码和作者单位等数据项，我们可以设想，如果验收文件中已经包含了这些数据项，是否存在编目系统的虚拟化问题，即编目完全融合在订购工作中，这是验收文件至少应该包含书名、作者名、出版单位、出版年份、单价、册数、订购部门/典藏部门代码等数据项。

阅读试题说明并结合流程图，可知出版单位在邮寄图书的同时附上到书清单和发票，发票上仅给出一份到书清单书的总册数和总金额。在处理 5 和处理 6 之后，同一张到书清单上的书必须分散在分类的到书文件中的不同位置上，可能会发生找不到与某一张发票相对应的图书的情况，从面导致发票上的金额与到书清单中的书价之和不一致时无法查对的后果。如果在到书清单中增加数据项"发票号"，则处理 7 中就能很容易地根据发票号找出与之相对应地所有图书。

根据试题说明，可知每种书都有一个书号，书号唯一地表示了一种图书。一种图书的不同复本的确良著录信息是相同的，但登录号对于每一册图书是唯一的，典藏部门代码和借出标志也是针对某一些册图书而有所不同的。所以，在书种文件中增加数据项"书号"后，复本文件中关于该种图书的出版发行信息及分类号、书名代码、作者代码等的数据项可以删除而不影响系统的功能。

【答案】

[问题 1] 书名、作者名、出版单位、出版年份、单价、册数、订购部门代码（或典藏部门代码）

[问题 2] 在到书清单中增加"发票号"

[问题 3] 书号、登录号、典藏部门代码、借出标志

6.11 1990 年度软件设计试题解析

1990 年度下午试题的软件分析试题涉及错误处理(试题 4 问题 1、试题 5 问题 1)、流程图修改及评估（试题 4 问题 2、问题 5 与试题 3 题问题 3）、处理的内容（试题 5 问题 2）和文件数据项构成（试题 5 问题 1）等 4 个方面的内容。总体来说这部分试题的难度较低，侧重能力才查。

3 道流程图试题中，一道是事务管理程序的流程图，一道是紧缩存储稀疏矩阵的转置，一道是探测法的实例。试题 1 与试题 3 的难度较之试题 2 相对低些，这主要是因为它们题材类型的差异与解答工作量的不同。试题 2 涉及的是一个相对复

杂抽象的问题，而且提供了两种算法，使解答题目的工作量大大增加。当然，解答这类题目的关键是切实把握试题说明和流程图的整体结构，这一点应当明确，应该注意到试题 3、度题 2 问题 3 对考生能力要求较高，要求考生能够迅速把握算法。

试题 1 （1990 年试题）

阅读下列说明和流程图，如图 6-60 所示，回答问题 1 至问题 3，把解答填入答卷的对应栏内。

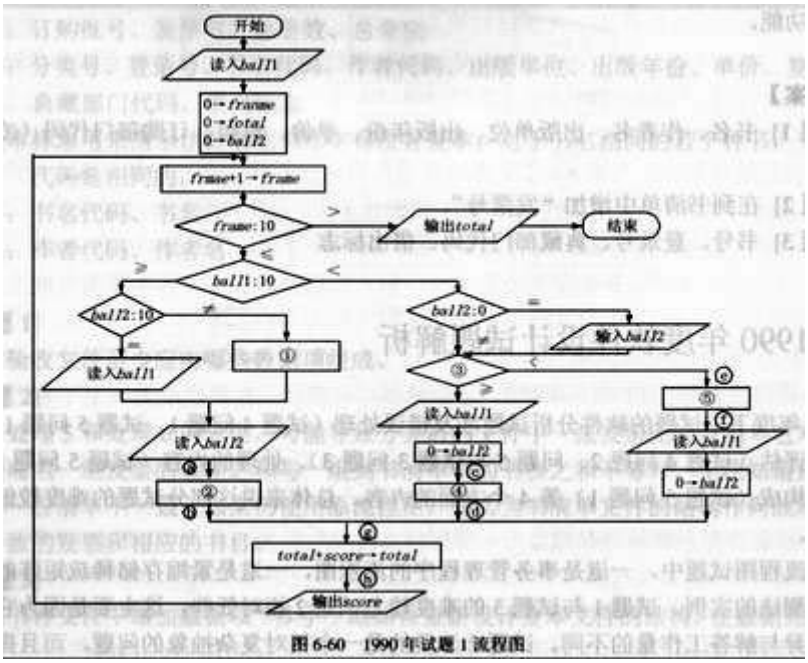


图 6-60 1990 年试题 1 流程图

[说明]
有一种游戏，是用滚球击 10 个性，比赛分 10 局，每局可滚球 1 次或多次，其规则和记分布方法如下：

（1）若一局的第一个球击倒全部 10 个柱（称为 strike），则这局不再滚球（对球 10 局来说，还可补滚两次球），其得分为 N 加下两次滚球所击倒的柱数。

（2）若一局数的第一个球未击倒 10 个柱，则可对剩下的柱再滚一次球。如果这局的两闪滚球击倒全部 10 个球（称为 spare），则这局不再滚球（对第 10 局来说，还可补滚一次球），其得分为 10 加下下一次滚球所击倒的柱数；否则，这局也不再滚球，其得分为本局两次滚球所击倒的柱数之和。

（3）总得分为 10 局得分之和。

流程图 6-60 读入每球击倒的倒数,计算产并输出每局得分 score 及总分 total 。图中 ball 和 ball2 分别存放每局第一个球和第二个球（如有的话）所击倒的柱数， frame 用于对局计数。

［问题 1］

填充流程图 6－60 中的圆圈，使之成为完整的流程图。

［问题 2］

若要把每球击倒的柱数记录在一个一维数组中，这个数组最少要有几个元素，最多要有几个元素。

［问题 3］

若计算每局得分的规则增加一条，当前面各局累积得分。超过 100 分时，每取得一次 strike 奖励 5 分，那么流程图 6-61 应插在流程图 6-60 中的(a)～(h)哪一个位置上。

[解析]

仔细阅读试题说明，不难发现流程图中有两个重要的判断应该注意。一是判断某局的第一次滚球结果是否为 strike ，即本次滚球击倒全部 10 个柱，二是在第一次滚球结果非 stike 时判断某局的前两次滚球是否将全部 10 个柱击倒。这两个判断在记分中有重要作用。

粗读流程图，会很快发现第一次滚球结果是否为 strike 的判断，而且流程图在此处分成两个分支，左边的分支处理第一次滚球结果为 strike 的情况，对应试题说明的（1）：右边的分支处理第一次滚球结果不是 srtike 的情况，对应试题说明的（2）。两个分支的会合处是 total+score→total 可以大致地判断这是将该局得分记入总分。

我们先看流程图右边的分支，该分支处理 strike 情况，从 ball 为第一次滚球得分来看，baal12 也应该是滚球得分。 判断 ball2 是否为 0 的作用何在？两个分支中都有这个判断，都是在 ball2 为 0 的时候读入 ball2，这里我们可以推断出 ball2 为 0 表示某次球是否滚过。在 ball2 不为 0 情况下，ball1 和 ball2 分别表示下两次滚球的得分。我们看到，在 ball2 是否为 0 的后面，一个分支在 ball2 为 0 的情况下读入 ball1,这说明另一个分支上的①应该是对 ball1 的操作。很明显，在 ball2 不为 0 的情况下，ball1（值为 10）已经失去了意义，应该用来记录下次的得分，所以①处应该填写 ball2→ball1。在 ball1 和 ball2 都准备就绪以后，按照流程图说明，"其得分为 10 加下两次滚球所击倒的柱数"，所以②处就应该填写 10＋ball1＋ball2→score

结合试题说明的（2），从 3 的判断处开始，流程图右边的分支又可分为两具小分支。从试题说明可以推断出判断的某局前两局滚球结果是否为 spare，即两次滚球的结果是否击倒全部 10 个柱，这个判断是容易构造的。借用以上的分析结果，在 ball1 和 ball2 准备就绪的情况下，只要判断 ball1 和 ball2 之和与 10 敦大即可，所以我们在 3 处填写 ball1+ball2:10 来完成这个判断，这样，从判断结果的两种不同情况可以把两个小分支的作用明确下来，即左边的小分支处理

结果为 spare 的情况，右边的上分支处理非结果 spare 的情况，结合试题说明与流程图结构，在这两个小分支里应该各有一个记分操作，否则在左右两个大分支的会合处 total+score→total 将不能正常工作，所以 4 处的答案应该是 10+ball1→score （"其得分为 10 加上下一次滚球所击倒的柱数"）5 处的答案应该是 ball1+ball2→score("其得分为本局两次滚球所击倒的柱灵敏之和")。

结合试题说明，不难看出，除第 10 局外，每局最多滚两次球（结果非 strike 时），最少滚 1 次球（结果为 strike 时）。这样记录前 9 局的得分在滚球次数最多的情况下需要一维数组有 18 个元素，在滚球次数最少的情况下需要一维数组有 9 个元素。第 10 局最少要滚两次球（结果非 strike 时），最多要滚 3 次球（结果为 Strike 时），这样，对一维数组的要求就是最少 11 个元素，最多 21 个元素。

问题 3 涉及的内容是在总分超过 100 时，每取得一次数 strike 奖励 5 分，因为这里是处理问题，结合以上对流程图整体结构的理解，我们就先排除了将所给流程图插入(c)处至(h)处的可能性，注意(g)和(h)的排除， 这里不详细说明排除的过程。②的解答我们已经给出，如果在之前插入给出的流程图，操作将会使奖励分丢失，权衡所有因素，给出的流程图应该插入②(b)处。

因为不是在考场上，我们再对流程图进行一点评论。本题目来判断某次球是否滚过。若某次滚球一无所获呢？如果用一个更为特殊的值。如负数、来表示某次球尚未滚过，就可以解决这个问题。

从上述的解析过程我们可以得出一点方法性的结论，即我们不一定能够对流程图的每一语句框都透彻理解，只要对流程图的结构有一个宏观把握，结合试题说明，那么解答问题的逻辑严密性和正确性还是保障的。如果忽视了对流程图整体结构的把握，一味地研究每一个语句框，将会陷于细微之处，最终是事倍功半。

【答案】

［问题 1］

①ball2→ball1 ②10＋ball1＋ball2→score ③ball1＋ball2：10
④10＋ball1→score ⑤ball1＋ball2→score

［问题 2］ 最少 11 个，最多 21 个

［问题 3］ 流程图 6－60 插在 (b) 点

试题 2 （1990 年试题 2）

阅读下列说明和流程图，如图 6－62 所示，回答问题 1 和问题 2。

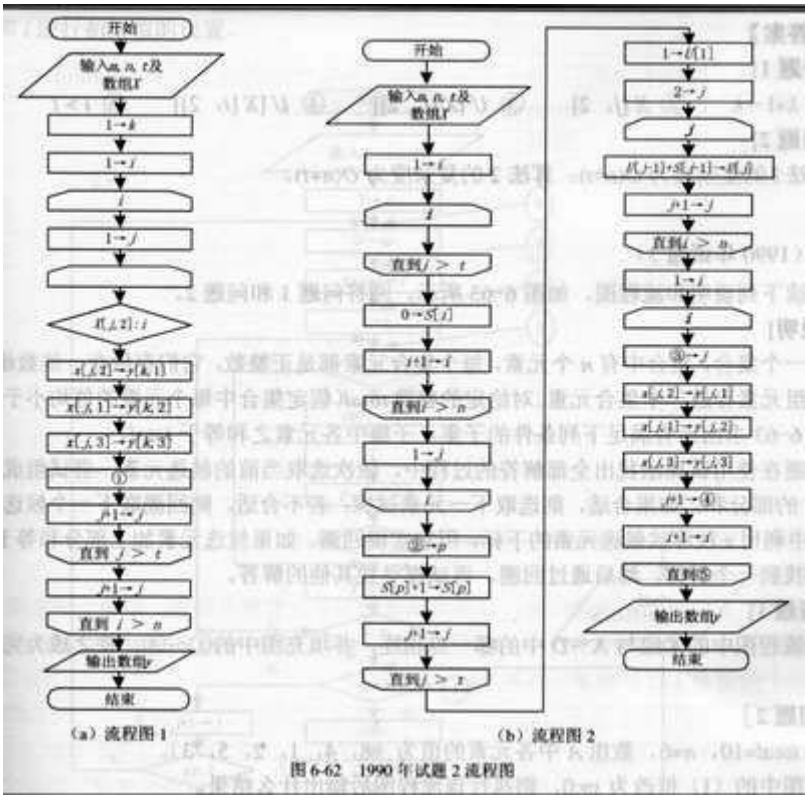


图6-62 1990年试题2流程图

[说明]

将一个 m × n 的矩阵 X 转置后存放到矩阵 Y 中，其计算复杂度为 O（m*n）。对稀疏矩阵来说，可以用紧凑的存储方式来减少所需的存储量，并降低计算复杂度。

已知一个有 t（t>0）个非零元素的 m×n 稀疏矩阵 W（每行每列至少有一个非零元素）以紧凑方式存放在数组 X[1：t,1:3]用中。X 中某行的 3 个值为 时表示在 W 的第 行第 列有一个非零元素 。假定 X 中的元素已按行号列号递增排序。现要求将 X 转置后以紧凑表示形式存放在数组 Y[1：t,1:3]中，并且 Y 也按行号列号递增排序。

下面描述了两种紧凑的稀疏矩阵的转置算法。

算法一见流程图 6－62 的图（a）。

算法二见流程图 6－62 的图(b)。图中，数组元素 S[i]用来存放 X 中列号为 的元素个数，数组元素 U[j]用来计算 X 中第 列元素在 Y 中的行号。

[问题 1]

填充流程图 6－62 的图（a）和图（b）中的①～⑤，使之实现相应的算法。

[问题 2]

分别写出算法一和算法二的计算复杂度。

【解析】

流程图 6－62 的图（a）和图（b）实现了两种将稀疏矩阵 X 转置后存放到 Y 中的算法，其中 X 和 Y 都是以三元组形式存储的稀疏矩阵，都按行号和列号递增排序。结合我们对稀疏矩阵知识的了解，三元组形式存储的稀疏矩阵的转置，实际上就是将三元组中的行号与列号对换。但本题的难度在于怎样使转置后的矩阵 Y 也按行号列号递增排序。

仔细阅读流程图 1，连续的 3 个语句框 X[j,2]→Y[k,1],X[j,1]→Y[k,2]和 X[j,3]→Y[k,3]揭示了本流程图的核心所在。我们可以从这 3 个语句框推断出本流程图所使用的算法，即依次将 X 中第 列的所有元素送到 Y 中作为 Y 的第 行的元素。显然，这里的 应该是递加的，所以我们在①处填写 。

流程图 6－62 的图（b）比较复杂，需要解答的问题也比较多。但粗读流程图之后，不难发现该流程图由 4 个循环组成，让我们通过对 4 个循环的解析来解答题目。

显然，第一个循环将一维数组 S 的所有元素（n 个）置 0。至于 S 的如何使用，我们先放在一边。I

由程序说明可知数组 S 中的元素 S[I]的用途是统计 X 中列号为 I 的元素个数，第 2 个循环中使用发数组 S。②将某个值赋给 p，从后面的使用来看 p 的值显然是一人列号。根据本循环执行 t 次的特点，我们推断②是斜某元素的列号（在循环中用 X[j,2]表示），赋给 P。

数组 U 中的元素的用途是统计 X 中第 j 列元素在 Y 中的行号。第 3 个循环执行了 n 次，循环结束后数组元素 U[j]指示了 Y 中第 j 行的第 1 个元素在 Y 中的超始位置。

考查第 4 个循环，该循环的作用是将 X 中所有三元组送到 Y 的相应元素中。结合以上对数组 U 的作用及元素值特点的了解,根据 X 元素中的列号(x[i,2])，在 U 中即可查找到该列元素在 Y 中的相应位置。所以在 3 和 4 处我们都填写 U[x[i,2]]。

本循环的作用已经明确，循环次数控制也就明确了，该循环的执行次数由 X 中三元组的个数决定。我们在 5 处填写，控制循环执行 t 次。

流程图 1 所描述的算法是由一个二重循环构成主体。其内循环次数为 t，外循环次数为 n。故二重循环的循环次数为 n*t。所以其计算复杂度为 0。

流程图 2 所描述的算法由 4 个单循环构成，它们的循环次数分别为 n、tn-1t,所以其计算复杂度为 0（n+t）。

[答案]

[问题 1]

①k+1→k ② X[j,2] ③ U[X[i,2]] ④ U[X[i,2]] ⑤i>t

[问题 2]

算法 1 的复杂度为 0(n*t);算法 1 的复杂度为 0(n+t).

试题 3 （1990 年试题 3）

阅读下列说明和流程图，如图 6-63 所示，回答问题 1 和问题 2。

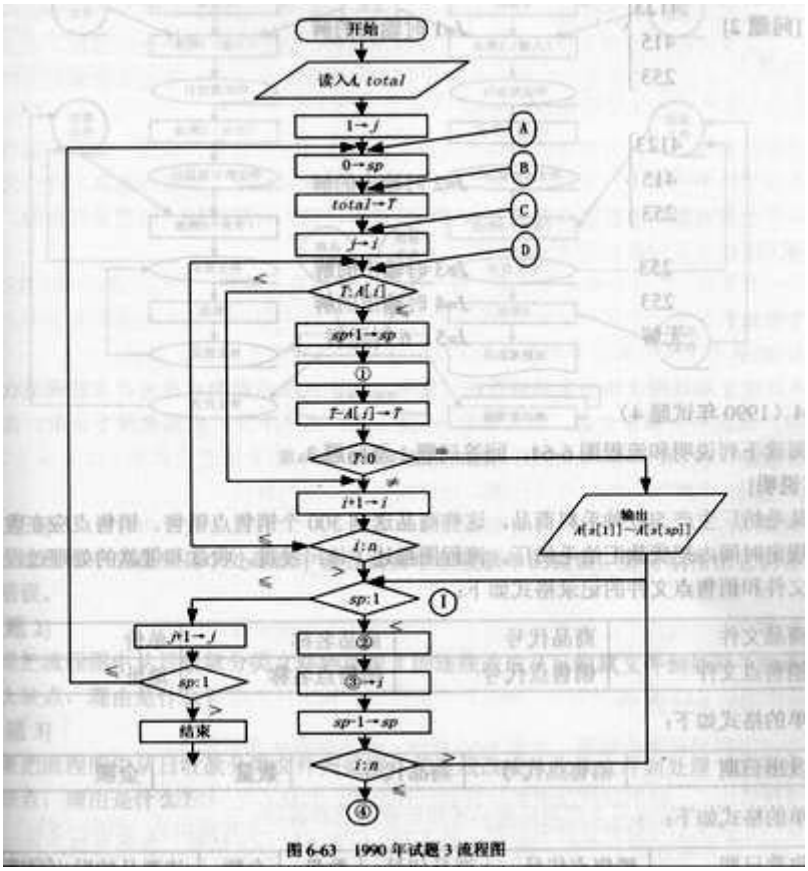


图 6-63 1990 年试题 3 流程图

[说明]

有一个集合，集合中有 n 个元素，每个集合元素都是正整数，它们存在一维数组 A 中，每个数组元素存放一个集合元素。对给定的整数 total（假定集合中每个元素的值均小于 total），流程图 6-63 求出所有满足下列条件的子集，子集中各元素之和等于 total。

本题在使用试探险去找出全部解答的过程中，依次选取当前的候选元素，尝试组成一个小于 total 的部分和。如果合适，则选取下一元素试探；若不合适，则回溯取下一人候选元素尝试。题中利用 S 栈存放候选元素的下标，用它实现回溯。如果候选元素加上部分和等于 total，则表示找到一个解答，然后通过回溯，再试探寻找其他的解答。

[问题 1]

问流程中的 4 应与 A～D 的哪一点相连，并填充图中的 1～3，使之成都市完整的流程图。

[问题 2]

设 total=10 n=6，数组 A 中元素的值为（8，4，1，2，。5，3）

若图中的(I)框改为 sp:0，则执行该流程图后输出什么结果。

[解析]

该题描述了一种用法试控法求解问题的处理流程，其核心是回溯技术。

我们先解决 4 与保处连接的问题，从 1→j, j→i, j+1→j 及两句 i:n 等语句来看，该程序的主体是一个二重循环，显然，4 是内循环的最后一句，应该连到内循环的第一句，结合我们的编辑经验，内循环的第一句不应在之前，因为我们看到变量 sp,T 和 i 都将在内循环中使用，每次内循环都要修改这些变量将使流程图的功能无法实现，所以 4 处应该连到 D 处。当然，这个判别断有很大的主观性，我们应以下的流程图中予以证实。

我们研究从 D 处以下到 T：0 的语句，由于 T 的初值已经是 total，从及此后的 T：0，这里应该是考虑某个元素是否包含进子集中，从试题说明中可以知道，考虑某一元素时，用堆栈 S 存放该元素的下标，显然，从来看，这里的元素下标就是 I，所以我们在处填写。

研究从 1 处到 4 处的内容，可以推断出这里处理一次探测到达集合最后一个元素及仍旧没有找到解而回溯的情况，回溯的方式是退栈，退栈的同时还要恢复 T 的值。然后从 s[sp]所指示的下一个元素寻找解答。从该部分语句的功能可以推出 2 和 3 的解答，即 T+A[s[sp]]→T 和 s[sp]+1。

结合以上对流程图的理解，我们印证了 4 处连到 D 处的正确性，因为每次外循环都要对 SP、T 和 i 进行初始值的设置。

解答问题 2 设计没有特殊的好方法，该题的解答是建立在对算法和流程图的正确理解之上的。根据流程图可知，流程图中每次执行内循环，都能找出 A[j]～A[n]中包含 A[j]且满足条件的解 sp:n 和 sp:1 的区别在于回溯的范围。若(I)处改为

sp:0。很明显，在 SP=1 时仍要回溯。即堆栈中只包含 A[j]时仍旧要回溯。所以，如果 sp:1 改为 sp:0,每次内循环都能找出 A[j]～A[n] 之间所有的解。

[答案]

[问题 1] ①$i \rightarrow s[sp]$　②$T + A[s[sp]] \rightarrow T$　③$s[sp]+1$　④D

[问题 2]

82 4123 415 253	J=1 时输出的解
4123 415 253	J=2 时输出的解
253	J=3 时输出的解
253	J=4 时输出的解
无解	J=5，6 时无解

试题 4 （1990 年试题 4）

阅读下列说明和流程图 6－64，回答问题 1 至问题 3。

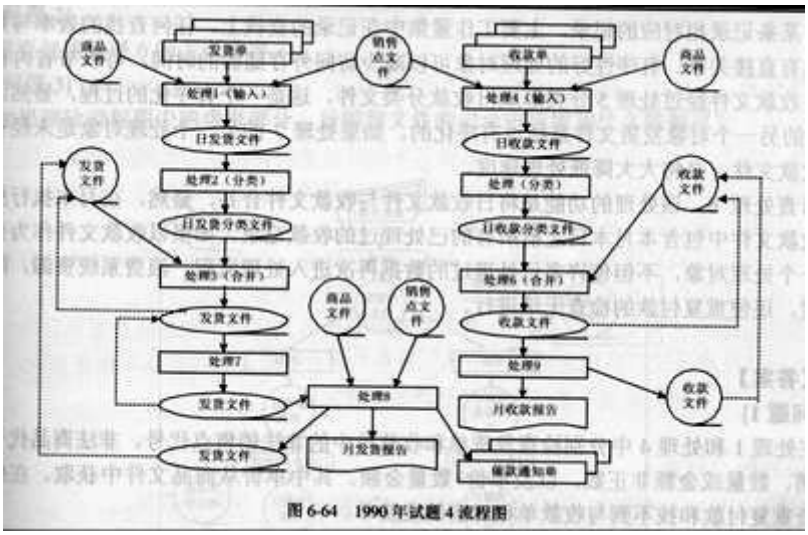


图 6-64 1990 年试题 4 流程图

[说明]

某毛纺厂生产 500 种毛料商品，这些商品送到 300 个销售点销售。销售点应在收到商品后的规定时间内把货款汇给毛纺厂。流程图描述了该厂发货、收款和催款的处理过程。其中商品文件和销售点文件的记录格式如下：

商品文件	商品代号	商品名称	单价
销售点文件	销售点代号	销售名称	地址

发货单的格式如下：

发出日期	销售点代号	商品代号	数量	金额

收款单的格式如下：

收款日期	销售点代号	商品代号	数量	金额	该商品的发出日期

处理 1～3 把当天的发货单合并到发货文件。处理 4～6 把当天的收款单合开到收款文件。每天在处理 3 和处理 5 做过之后，由处理 7 在发货文件中当天已收款的记录上加上已收款标记，处理 8 在月末执行一次，它有 3 个功能（1）汇总输出本月发货清单；（2）删除发货文件中已收款的所有记录，形成一个新的发货文件，作为下月初处理时的初始文件；（3）产生催款能知单，以便对那些一个月以前已发货但至今仍未收到货款的售点催款。处理 9 也每月末执行一次，除输出本月收款报告外，还删除收款文件中的所有记录。现假定不会有完全要同的发货单。

［问题 1］

指出流程图中应在哪几个处理框中检查发货单和收款单的错误，并分别指出它们各能指出什么错误。

［问题 2］

如果把流程图中从日收款分类文件到处理 7 的连线改成从日收款文件到处理 7 的连线，则有什么缺点，理由是什么？

［问题 3］

如果把流程图中从日收款分类文件到处理 7 的连线改成从收款文件到处理 7 的连线，则有什么缺点，理由是什么？

【解析】

阅读试题说明的目的是理解流程图的功能、某些文件的结构和某些处理的功能，这些内容为我们解答问题提供了丰富而有效的信息，不可忽视。

问题 1 要求我们回答在检查何处检查发货单与收款单中的错误，能够检查出什么错误。首先，检查数据的错误应该是数据进入系统后的第一道手续，否则，就不能保证后面的处理使用的是正确的数据。我们可以看到，处理 1 是输入发货单，处理 4 是输入收款单，则必须在处理 1 检查发货单的错误，在处理 4 检查收款单

的错误。一般来说，错误检查应该考虑错误的客观存在而不能考虑错误的主观不存在，输入的任何数据项都存在出错的可能性，因此我们可能检查到非法的销售代号、非法的商品代号、非法的日期、非法的数量和非法的金额。由于数量和金额具有依赖关系，因此数量价格（从商品文件取得）金额也是可能检查到错误，另外，处理 7 要将收款单和发货单对应起来，我们不能排除重复付款和找不到与收款单对应的发货单的可能性。

我们首先考查处理 7 的功能，因为问题 2 和问题 3 都是围绕处理 7 出现的。处理 7 在发货文件中当天已收款的记录上加已收款标记，其内部处理应该是在发货文件中查找出与收款文件中某条记录相对应的记录，主要工作量集中在记录的查找上，任何查找的效率与数据的有序性有直接关系。有序性好的处理对象可以减少访问外存储器的时间，可以节省内存。

日收款文件经过处理 5 分类形成日收款分类文件，这是一个有序化的过程，显然，处理 7 处理的另一个对象发货文件提经过有序化的。如果处理 7 的另一个处理对象是未经有充化的日收款文件，必将大大降低处理速度。

考查处理 6，该处理的功能是将日收款文件与收款文件合并，显然，在月末执行处理 9 之前收款文件中包含本月本日之前所有的已处理过的收款记录。如果以收款文件作为处理 7 的另一个处理对象，不但使许多已处理过的数据再次进入处理流程，浪费系统资源，降低处理速度，还使重复付款的检查无法进行。

【答案】

［问题 1］

在处理 1 和处理 4 中分别检查发货单和收款单中的非法销售点代号、非法商品代号、非法日期、数量或金额非正数，以及单价＊数量金额。其中单价从商品文件中获取。在处理 7 中检查重复付款和找不到与收款单相对应的发货单。

［问题 2］

因为文件未分类，所以处理时要增加访问外存的时间或需要大量内存，从而降低了处理速度。

［问题 3］

收款文件中包含了已处理过的记录，从而降低了处理速度。

试题 5 （1990 年试题 5）

阅读下列说明和流程图，如图 6-65 所示，回答问题 1 至问题 3。

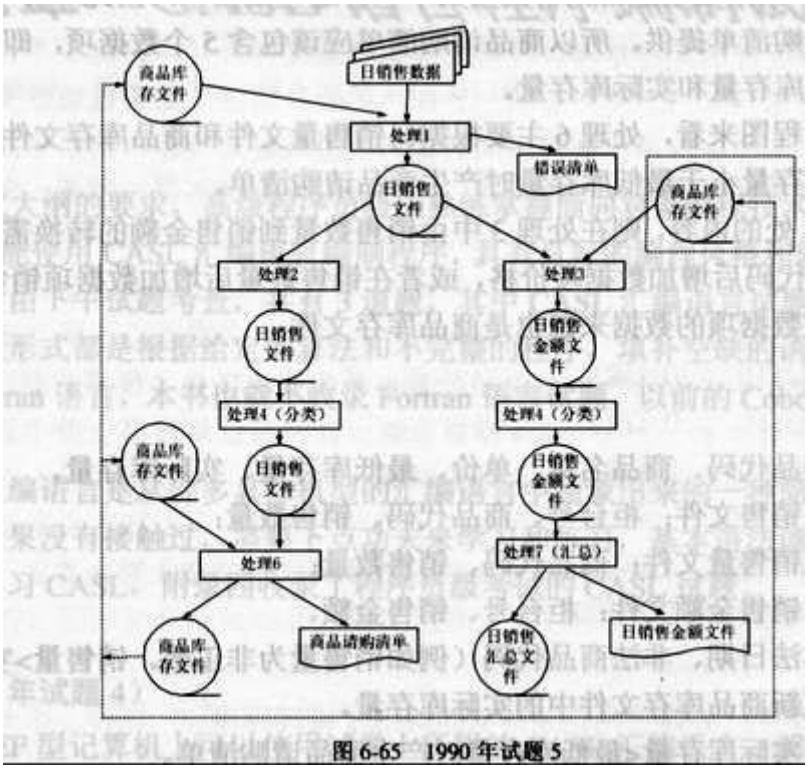


图 6-65 1990 年试题 5

［说明］

（1）流程图描述某大型商店商品销售的数据处理流程。

（2）商店设有若干柜台，同一种商品可能在几个柜台上销售，各柜台每天提供一组日销售数据，其格式如下：

日期、柜台号、商品代码、销售数量、商品代码、销售数量……

（3）数据处理系统每日产生一份反映各柜台当日销售金额和商店日销售金额的"日销售金额报告"，必要时还产生一份"商品请购报告"，给出那些低于最低库存量的商品代码、商品名称、最低库存量和实际库存量。处理过程中产生存档的"日销售文件"和临时工作文件"日销售文件"和"日销售金额文件"。

（4）系统中所用到的数据均来自数据文件。

（5）流程图中的商品库存文件的记录已按关键字"商品代码"排序。

［问题 1］

①指出商品库存文件的记录中必须包括哪些数据项？

②分别指出在日销售文件，日销售量文件和日销售金额文件的记录中至少应包括哪些数据项，同时不产生数据冗余？

③错误清单可能指出哪些错误？

［问题 2］

简要叙述处理 6 的主要内容。

［问题 3］

如果删除流程图中的虚框部分，日销售文件的记录中应增加什么数据项？

【解析】

本题描述的是商品销售的数据处理流程。阅读试题说明，注意说明中的（3）和（4），对系统的功能和数据来源有一个大体把握。粗读流程图在处理 1 产生日销售文件后大致分为两个分支，左边的的分支处理日销售量及商品请购事务，右边的分支处理销售金额事务。其中在处理 1、处理 3 和处理 6 都要用到商品库存文件。

从流程图的处理 1 开始读起，处理 1 的作用输入日销售数据，生成日销售文件和错误清单。问题 1 的问错误清单可能指出哪些错误，答案的范围是明确的。凡是输入的日销售数据都有可能发生错误，如非法日期、非法商品代码（商品库存文件中无对应记录）和非法销售数量。

处理 1 通过输入日销售数据，与商品库存文件相配合，产生日销售文件。除日期外，日销售数据包括柜台号、商品代码和销售数量。如果商品库存文件中的数据项在这里已经包括进日销售文件，那么在处理 3 和处理 6 处应该不再用到商品库存文件，所以我们断定日销售文件的数据全部严自日销售数据，即它最多包括柜台号、商品代码和销售数量 3 个部分。考查这 3 个数据项以后的用途，未发现数据冗余。

日销售文件通过处理 2 和处理 3 生成日销售量文件和日销售金额文件，其中日销售量文件用作销售量处理，因此至少应该包括商品代码和销倍数量。日销售金额文件要产生反映各柜台当日销售金额和商店日销售金额的日销售数量，日销售金额文件要产生反映各柜台当日销售金额和商店日销售金额报告，所以至少应该包括柜台号和日销售金额。数据项销售金额可以借助商品库存文件得到。

商品库存文件必须包含哪些数据项？首先是试题说明中提到的关键字商品代码；其次，在处理 3 中，商品销售数量到销售金额的转换需要商品价格，该数据项也必须包含在商品库存文件中；第三，处理 6 根据日销售量文件和商品库存文件产生商品请购清单，商品请购清单中的商品名称、最低库存量和实际存量 3 个数据项尚未落实来源。由于所有数据都来自数据文件，而另一个原始数据源日销售数据又不包含这些数据项，因此综合考虑，这 3 个数据项也应该由商品请购清单提供。所以商品请购单应该包含 5 个数据项，即商品代码、商品名称、价格、最低库存量和实际库存量。

从试题说明和流程图来看，处理 6 主要根据日销售文件和商品库存文件来更新商品库存文件，并在实际库存量小于最低库存量产生商品请购清单。

如果删除虚框 A 处的内容，则在处理 3 中由销售数量到销售金额的转换需要日销售文件的支持，应该在商品代码后增加数据项价格，或者在销售数量后增加数据项销售金额。当然，在日销售文件中增加数据项来源也是商品库存文件。

【答案】

［问题 1］①商品代码、商品名称、单价、最低库存量、实际库存量。

②日销售文件：柜台号、商品代码、销售数量；
日销售金额文件：柜台号、销售金额。

③非法日期、非法商品代码（例如销售量为非正数，销售量＞实际库量）。

［问题 2］ ①更新商品库存文件中的实际库存量。

②当实际库存量＜最低库存量时产生商品请清单。

［问题 3］ 在日销售文件中增加"单价"或"金额"。

6. CASL 语言程序编制试题精解

第七章　CASL 语言程序编制度量精解

根据考试大纲的栺，高级程序员级应熟练掌握面向对象编程技术，熟练使用 C/C＋＋语言编制程序，能使用 CASL 汇编语言编制程序，具有良好的编程风格和较强的算法设计能力。程序编制能力由下午度量考查，共有 3 道题，其中 CASL 汇编语言试题 1 道，C 语言程序试题 2 道，试题形式都是根据给定的算法和不完整的程序，填补空缺的语句。从 1999 年开始，不再选考 Fortran 语言，本书也就不收录 Fortran 语言试题，以前的 Cobol 、Pascal 试题也没有收录。

CASL 汇编语言是从许多具体机型的汇编语言中抽象出来的一种简化的、专用于考试的汇编语言。如果没有接触过，需要下点功夫来学习和练习，基本语法请参见本书的附录三。为便于考生复习 CASL，附录四收录了程序员级考试的试题。

试题 1　（2000 年试题 4）

在 COMEF 型计算机上可以使用试卷上所附的 CASL 汇编语言。程序说明和 CASL 程序，将应填入 （n） 处的字句，写在答卷的对应栏内。

[程序 1 说明]

1. 本子程序根据每位职工的基本工资（非负值）和他完成产品的超额数或不足数计算职工的应发工资。

2. 主程序调用时，CRI 中给出子程序所需参数的起始地址，参数的存放次序如下表。

GR1

a1
b1
c1
a2
b2
c2
…
an
bn
cn
-1(结束标志)

其中为职工的基本工资，为职工的完成产品的超额数或不足数，为职工的应发工资数（i=1,2,…,n）。

bi 以原码形式存放（大于零为超额，小于零为不足）。基本工资与计算所得的应发工资以补码形式存放。

3. 应发工资的计算规则为：

• 恰好完成定额数（此时为零），应发工资即为基本工资。

• 每超额 4 件，在基本工资基础上增加 10 元（不到 4 件，以 4 计算。例如超额数为 10 时，增加 30 元）。

• 每不足 4 件，在基本工资基础上减 5 元（不到 4 个，以 4 计算。例如，不足数为 5 时，减 10 元）。

[程序 1]

```
        START
```

```
BEG     PUSH 0,GR1
        PUSH 0,GR1
L1      PUSH 0.GR2
        PUSH 0.GR3
        (1)
        JEA GR0,0,GR2
        JMI FINISH
        LD GR3,3,GR1
        LEA GR2,0,GR3
        AND GR2,C7FFF
        JZE L3
L2      SRL GR3,15
        LEA GR2.-1,GR2
        (2)
L3      LEA GR2,-4,GR2
        JPZ L2
        (3)
FINISH  (4)
        (5)
        POP GR3
        POP GR2
7FFF    POP GR1
BONUS   RET
        DC #7FFF
        DC 10
        DC -5
        END
```

[解析]

阅读程序说明，特别要注意在 GRI 中给出的子程序所需要的参数。可以看出，有所给参数中包含 4 类数据：1、ai, ，职工 i 的基本工资；2、bi ，职工 i 的完成听超额数或不足数；3、ci，职工 i 的应发工资；4、－1，结束标志。

在阅读程序说明的其他部分以后，开始阅读程序。从程序开始到第 4 行是子程序的起始部分，执行保存现场的操作，将 GR1、GR2、GR3 的内容分别入栈，这一部分是容易理解的，而且也与本题的解答关系不大。

第 5 行需要填写内容，但现在程序刚刚开始，我们很难从已知的信息推断出这里应该填写的内容。那么就跳过第 5 行继续阅读程序。第 6 行的语句给我们很多想像。在这里将 GR2 的内容取到 GR0 中去了。注意，子程序开始时，只有 GR1 的内容对子程序是有用的，GR2 的内容是由调用子程序时主程序的运行善的。如果第 5 行不是对 GR2 进行操作，将某数据装入 GR2，第 6 行的操作将导致无法预测的结果，尤其时在第 7 行有一个 JMI 语句。这样，我们就可以判断第 5 行的大体内

容。即在 GR2 中装入某个有用的数据。倡这里装入的是什么数据呢：仍旧需要结合后面的语句进行解答。

第 7 行的"JMI FINISH "可能导致程序的结束，问题是什么时候程序可能结束。程序当然是在遇到结束标志时结束，但在程序的开始部分就如此安排显然时不妥当的，我们再考察数据的存储。可能出现负数的只有结束标志和两类，显然遇到数额不足的职工就退出程序也是不妥当的。仍旧要从结束标志上看。现在让我们来假设，如果此处装入 GR2 的内容是 GRI 所指地址的内容，那么在处理完一个职工的工资后将 GRI 内容后跳 3 个字节，在处理完 了所有职工后，GR1 所指地址的内容巍然是结束标志。暂且作这样的假设，在填空（1）中填写"LD GR2,0，GR1"。

继续阅读程序。第 8 行的语句告诉我们将某位职工的超额数或不足数装入 GR3，然后在第 9 行中又把 GR3 的内容拷贝到 GR2 中。第 10 行的 AND 操作究竟有什么作用？如果我们注意到是以原码方式存储的，而且注意到数据 C7FF 的特点，就不难判断这是取超额数或不足数的绝对值。当绝对值为 0 时就跳转到第 17 行的 L3。那么，从第 17 行起，必定有一个存储工资额的操作，因为在超额数或不足数为 0 时，应发工资即为基本工资。

接着阅读第 12 行。第 12 行的逻辑右移操作是简单的，可是目的难以判断。暂且在草稿纸上演示一番。当 GR3 是超额数时，经过右移，GR3 的内容就变成 0；当 GR3 是不足数时，经过右移，GR3 的内容就变成 1。这 1 和 0 怎样表示是超额数还是不足数呢？先跳过再说。第 13 行的操作更让人摸不着头脑，为什么把数额给减了一个去，我们也暂时存疑吧。

第 14 行是需要填写内容的，从下面的将 GR2 内容减 4 看，这里是将超额数或不足数的绝对值减了 4，结合后面的"JPZ L2"，第 14 行的语句应该是将奖金或罚款与基础工资进行加减操作的语句。考察变量 BONUS 定义，如果直接引用该变量，那么它就是 10，如果引用该变量的下一个存储单元，就是－5。让我们再次提起第 12 行的逻辑右移。很、明显，如果我们这里使用 GR3 的内容来处理奖金或罚款，应该是很方便的。比如说，在这里填写"ADD GR0，BONUS，GR3"，这样，当 GR3 内容为 0（数额为超额数）时，这里回到 GR0 中的就是 10，当 GR3 内容为 1 时（数额为不足数）时，这里加到 GR0 中的就是－5。再仔细阅读从第 12 行到第 16 行的语句，这样的实现是能够满足需要的。

现在看第 16 行，JPZ 操作是在等于或大于时跳转。我们在这里可以举一个例子，当超额数为 4 时，第一次执行第 14、15 行时就完成了对应发工资的计算和对超额数进行减 4 操作，接着执行第 16 行，不免要跳转到第 14 行再进行一次奖金发放和超额数减 4，然后才能够执行第 17 行的操作。显然这是不符合道理的。结合第 13 行考虑，上述的错误就不会发生了，具体过程学员可以自己计算。

关于第 17 行的内容，在前面我们已经判断为存储应发工资额了。现在关键是看如何实现，由于是要写到某个内存地址，所以必须使用 ST。结合以上程序，可以知道应发工资额存储在 GR0 中，那么这里的第一个操作数就是 GR0 了。如何定位地址？程序中始终使用 GR1 来指示内存地址，这里最好不要例外，而且要想用

别的寄存器指示内存地址还需要额外的操作，在这里是不允许的，所以我们就这样构造答案："ST GR0，1，GR1"

在第 18、19 行是在处理完一个职工的信息之后的操作，这里应该是什么操作？显然，应该是将 GR1 的内容加 3，为下一个职工的信息处理提供正确的内存地址信息，然后再强制跳转到 L1 处。
在本题的答案中，我们应注意两点，一是始终注意寄存器内容的变化，二是始终注意变量的灵活使用。注意寄存器内容的变化，不仅要求掌握寄存器的内容发生了哪些变化，而且要求对寄存器在某种情况下应该如何变化有清醒的把握。对变量的灵活使用也包括两个方面，一是明确变量使用后发生的效果，其次是如何对连续存储的变量进行操作。

[答案]

（1）LD GR2，0，GR1

（2）ADD GR0，BONUS，GR3

（3）ST GR0，2，GR1

（4）LEA GR1，3，GR1

（5）JMP L1

试题 2 （1999 年试题 4）

在 COMST 型计算机上可以使用试卷上所付的 CASL 汇编语言，阅读程序说明和 CASL 程序，把应填入（n）处的字句，写在答卷的对应栏内。

[程序 1 说明]

本子程序是对 15 位二进位串，求它的奇校验位，生成 16 位二进位串，使 16 位二进位串中有奇数个 1。

进入此子程序时，15 位二进位串在 GR1 的第 1 位至 15 位，并假定 GR1 的第 0 位是 0。求得的奇校验位装配在 GR1 的第 0 位上。

[程序 1]

```
START
BEG     PUSH 0，GR2
        PUSH 0，GR3
        LEA GR3，1
```

```
        （1）
L1      SLL GR2，1
        （2）
        LEA GR3，1，GR3
L2      JZE L3
        JMP L1
        （3）
        ST GR3，WORK
        ADD GR1，WORK
        POP GR3
        POP GR2
        RET
WORK    END 1
```

[程序 2 说明]

子程序 SUM 是将存储字 A 起的 n(n>0)个字求和，并将结果存于存储字 B 中。

调用该子程序时，主程序在 GR1 中给出存放子程序所需参数的起始地址。参数的存放次序如下：

（GR1）	+0	A
	+1	n
	+2	B

[程序 2]

```
SUM     START
        LD GR2，0，GR1
L5      LD GR3，1，GR1
        LEA GR0，0
        ADD GR0，0，GR2
        LEA GR2，1，GR2
        （4）
        JNZ L5
        （5）
        ST GR0，0，GR3
        RET
        END
```

[解析]

[程序 1]

在解答试题之前我们在这里引用 1992 年程序员级考试 CASL 语言试题的第一道题。试题说明大同小异，只是现在要求的是奇校验，而 1992 年要求的是偶校验而已，我们只能说，两道试题不仅仅是相似。下面看 1992 年的试题：

```
START
BEG       （1）
          （2）
LEA       GR3，0
LEA       GR2，0，GR1
          （3）
          （4）
LEA       GR3，1，GR3
L2        （5）
JMP       L1
L3        （6）
ST        GR3，WORK
ADD       GR1，WORK
POP       GR3
POP       GR2
RET
WORK      DS 1
          END
```

首先阅读试题说明和程序 1。试题说明没有提供更多的关于算法的信息。但既然是求奇偶校验位，常用的方法是移位并逐位确认并累积"1"的个数，从程序中的"SSL GR2，1"与"LEA GR3，1，GR3"两条语句可以看到这种算法的轮廓。下面我们通过读程序来解答题目。

在填空（1）之前是一些程序初始化工作，从程序的开头和结尾处对 GR2 和 GR3 的 POP 和 PUSH 操作来看，前 3 条语句只是设置了 GR3 的初值，而填空（1）之后马上又对 GR2 进行操作，显而易见填空（1）是对 GR2 的赋初值进行操作。从程序中的"SLL GR2，1"和"ADD GR1，WORK"可以看出，真正在程序中进行移位比较的是 GR2，求校验位的 15 位二进制位串还保存在 GR1 中，没有参与移位运算。基于上述分析，（1）的内容就很明确了，即"LEA GR2，0，GR1"，完成取待处理的二进制位串道 GR2。

从标号 L1 一直到语句"JMP L1"可以看出这是一个循环，那么作为程序中唯一的循环结构，确定"1"的个数应该是在这个循环结构中完成的。"SLL GR2，1"和"LEA GR3，1，GR3"增强了我们对这个判断的饿信心。不管 GR3 的初值如何，在循环体中必须有对当前是否为"1"的判断，综合考虑循环体，任务只能由语句（2）来完成。那么，这里应该是判断其为"0"呢，还是判断其为"1"？跳转的目的地是哪里？

假设跳转到 L2，那么我们看在这里如果 GR2 内容全为"0"的情况下要跳转到 L3。如果 GR2 的内容全为"0"，则表示移位操作已经完成，程序也就应该结束了。跳

转到 L2 的假设使跳转到 L3 的假设失去了意义，因为只有程序计算出二进制位串中"1"的个数后才能跳转到 L3。由此可以断定跳转的目的地是 L2。

但我们仍旧不能确定填空（2）中的判断。我们假设是判断"1"，那么在当前位为"0"的情况下执行"LEA GR3，1，GR3"，累加"1"的个数。在全"0"的情况下也不会发生跳转，这样就多加了一个数位。如果判断填空（3）是对"0"的个数的处理，填空（3）和后续的两句实现写入校验位是不可能的，所以这里应该是判断"0"的。综合考虑，我们就填写"JPZ L2"。

现在通读程序，不难得出结论，当程序运行到 L3 时，GR3 的内容为二进制位串中"1"的个数加 1。如何使 GR3 的值满足"ST GR3，WORK"和"ADD GR1，WORK"的要求，共同完成写二进制位串的校验位的功能？首先，从上面提到的两句来看是在 GR3 中形成了一个首位为"1"或"0"（依"1"的个数而定），其余各位全部是"0"的数字。我们看，由于 GR3 的初值是 1，当二进制位串中有偶数个"1"时，GR3 中即为奇数，反之为偶数。根据奇校验的规则，当二进制位串中有奇数个"1"时保持首位为"0"，反之则将其首位变为"1"。考虑到当 GR3 内容为偶数时末位为"0"为奇数时末位为"1"，我们可以在填空（3）中填写"SLL GR3，15"来使 GR3 的末位移动到首位，而其余各位都是"0"。

[程序 2]

鉴于这是一道非常容易的试题，我们直接通过阅读程序来解答。

首先，程序的开头是两个 LD 语句 ，结合程序说明，很容易确定它们的含义，即将起始地址 A 和数字的个数分别读进 GR2 和 GR3。从此后对 GR0 的操作来看，GR0 中存储的是加法运算的结果，而语句"LEA　GR2，　1，　GR2"则实现地址指针的偏移。

那么填空（4）中的语句究竟应该是什么呢？从下面的"JNZ　L5"来看，L5 和填空（4）之间的语句构成一个完整的循环体。从整体上考虑，（4）应该是循环控制变量的修改，由于 GR2 和 GR1 都不能用作循环控制条件，结合 GR3 内容的含义，我们构造"LEA　GR3，　－1，　GR3"完成循环变量的修改。

而"JNZ　L5"之后的部分，跳过填空（5），"ST　GR0,0，GR3"是将求和运算的结果存储到 GR3 所指示的地址中去。从程序说明可知 GR3 所指示的地址应该是(GR1＋2)，那么（5）的内容就很容易确定了，即"LD　GR3,2，GR1"，完成将 B 传入 GR3 的操作。

在解答上面两道题的过程中，我们有 3 点体会：1、研究历年试题很有用；2、注意 LD 和 LEA；3、注意通过条件跳转构造循环的算法。

[答案]

（1） LEA　GR2,0，GR1

（2） JPZ L2

（3） SLL GR3,15

（4） LEA GR3，－1，GR3

（5） LD GR3,2，GR1

试题 3 （1998 年试题 4）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读以下程序说明和 CASL 程序，将应填入（n）处的字句，写在答卷的对应栏内。

[程序说明]

本程序是统计字符串中数字字符'0'至'9'的出现次数。

字符串中的每个字符是用 ASCII 码存储。一个存储单元存放两个字符，每个字符占 8 位二进位。
程序中，被统计的字符串从左至右存放在 STR 开始的连续单元中，并假定其长度不超过 200，字符串以'.'符作为结束。NCH 开始的 10 个单元存放统计结果。

[程序]

```
        START MIN
MIN     LEA GR2,9
        LEA GR0,0
L1      (1)
        LEA GR2,-1,GR2
        JPZ L1
        LEA GR4,0
        LEA GR1.0
L2      LD GR2,STR,GR1
        EOR GR4,C1
        JNZ RL
        (2)
PL      STL CR2,8
        LEA GR3,0,GR2
        SUB GR3,C9
        JMI L3
        JNZ L4
L3      LEA GR3,0,GR2
```

```
          SUB GR3,C0
          JMI L5
          LEA GR2,1
          (3)
          (4)
L4        LEA GR4,0,GR4
          JNZ L2
          (5)
          JMP L2
L5        SUB GR2,C
          JNZ L4
          EXIT
C1        DC 1
C         DC '.'
C0        DC '0'
C9        DC '9'
STR       DS 200
NCH       DS 10
          END
```

[解析]

仔细阅读程序说明，不难发现本题的难度在于一个存储单元存放两个字符，这样每个存储单元必须进行再次统计。如何控制先统计存储单元的前 8 位再统计其后 8 位？让我们带着这个问题来阅读程序，通过对程序的理解来集会其控制算法。

在通读程序之后 ，我们会发现一直到"JPZ L1"之前的语句无论在位置上还是在逻辑上都属程序的前奏部分。因为此后没有出现跳转到 L1 的语句。从"LEA GR1,0"、"L2 LD GR2 STR，GR1"大致可以看出 GR1 是用作统计时的地址指针的，不可能再次出现 GR1 置 0 的情况，因此排队了后面 4 个空语句中存在跳转到 L1 的可能性，确定了该部分语句作为程序前奏的地位。通过"MIN LEA GR2,9"和"LEA GR2，－1，GR2"及"JPZ L1"可以断定此处语句共执行了 10 次。程序中需要进行 0 次操作的变量只有 NCH 一个，所以我们就可以大胆的在填空（1）中填写"ST GR0，NCH，GR2"，需要注意的是，这里 GR2 既作循环计数器又作地址偏移指针。

此后开始的语句应该是本程序要完成的中心任务，一一统计各字符出现的次数。为了在理解算法的时候不出现偏差，应先通过阅读程序来确定各个寄存器在程序中的用途。GR1 在程序中出现过两次，而且是置 0 操作和用作取内存内容的地址偏移指针，不存在地址偏移指针固定不变的情况，由此可以断定在后面的 4 个填空处至少应该有一处是对 GR1 进行操作的，只是位置有待确定而已。GR4 在程序中出现了 3 次，分别是置 0、与 C1 导电异或和检测是否不为 0，考虑到异或的结果是使 GR4 的内容在 1 和 0 之间变换，我们大致可以判定 GR4 是用作某种标志值。至于 GR2、GR3 的用途，由于出现次数较多，用法灵活，因此只能在程序中结合具体情况进行分析。

从"L2 LD GR2，STR，GR1"开始阅读程序，因为已经大致判定 GR4 是用途某种标志的，所以可以发现，在"DOR GR4，C1"之后有一个跳转，即当 GR4 内容不为 0 时跳转到"RL SRL GR2,8"而 GR2 的内容是 STR 某一内存单元的值。问题就在这里，程序是否通过 GR4 的值来控制统计前 8 位还是后 8 位？如果是这样，那么在不发生跳转时，从上下语句来看，能够看到当处理后 8 位时不发生跳转。怎样保证右移 8 位之后 GR2 的内容正好是 STR 某一内存单元后 8 位的值？方法只有一个，就是先把 GR2 左移 8 位。因为仅需要 8 个二进制位的值，所以防止移位的时候出现符号位问题，结合语句"SRL GR2,8"我们选择在这里填写语句"SLL GR2,8"。

继续跟踪 GR4 的变化，在语句"L4 LEA GR4,0，GR4"和"JNZ L2"处，可以结合以上的分析结果断定此处是判别一个内存单元是否已经处理完毕。如果已经处理完毕，很明显应该进行偏移地址的递加，应该是对 GR1 的操作，于是填空（5）处就填写有关 GR1 加 1 的语句，这里语句是不唯一的，可以填写"LEA GR1,1，GR1"，这是比较平实的方案。如果考虑到物尽其用的原则，不妨填写"ADD GR1，C1"的语句，充分发挥 C1 的作用，这已经是个人习惯的问题了。

以上分析并未涉及比较统计操作，只是对程序中的特殊用途寄存器进行分析，判断出填空（2）和（5）的内容。现在尚未处理的两个空白应该是涉及到比较和计数两方面操作的。

阅读程序，不难发现当程序执行到"L3 LEA GR3,0，GR2"处时，已经表明 GR2（GR3）的内容是一个小于"9"（C9）的字符。执行到"LEA GR2,1" 处，各寄存器的内容应该是：GR3－－－－统计比较对象（应该是大于"0"）减去"0"（C0）的值，GR2－－预备进行计数累加的数值 1。那么，在下面的两个空白处应该填写的内容就是计算累加的工作，结合对程序整体的理解，我们就在这里填写"ADD GR2，NCH，GR3"与"ST GR2，NCH ，GR3"两个语句。注意 GR3 的内容既然是统计比较对象（应该是大于"0"）减去"0"（C0）的值。也就应该是进行计数时相对于 NCH 首地址的偏移量。

[答案]

(1) ST GR0,NCH,GR2

(2) SLL GR2,8

(3) ADD GR2,NCH,GR3

(4) ST GR2,NCH, GR3

(5) LEA GR1,1,GR1 或 ADD GR1,C1

试题 4 （1997 年试题 4）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读下列程序说明和 CASL 程序，将应填入（n）处的字句，写在答卷的对应栏内。

[程序说明]

本子程序将一个非负二进整数翻译成 5 位十进整数字符。

进入子程序时，在 GR0 中给出被　翻译的非负二进整数，在 GR2 中给出存放 5 位十进整数字字符的起始地址。

十进制数字符字用 ASCII 码表示。当结果小于位时，左边无意义的 0 用空白符替换；当二进整数为 0 时，在（GR2）＋4 中存放 0 的 ASCII 码。

数字字符 0 至 9 的 ASCII 码是 48 至 57，空白符的 ASCII 码是 32。

[程序]

```
          START
          LEA GR1,0
          LEA GR3,32
L1        (1)
          JPZ L2
          ST GR3,0,GR2
          LEA GR2,1,GR2
          LEA GR1,1, GR1
          LEA GR4,-4, GR1
          JNZ L1
L2        (2)
L3        (3)
          JMI L4
          SUB GR0,SN0, GR1
          LEA GR3,1, GR3
          (4)
L4        ST GR3,0, GR2
          LEA GR2,1, GR2
          LEA GR1, GR1
          (5)
          JNZ L2
          RET
SN0       DC 10000
          DC 1000
          DC 100
          DC 10
          DC 1
          END
```

[解析]

本题所给出的程序是对数进行形式转换，将非负二进整数转换成由 ASCII 码表示的十进制整数字符串，这里我们首选要的。另外在 CASL 语言中没有除法指令，要利用减法来实现除法。求商的过程是将被除数减去除数，得到的差再减除数，这样一直到不够减为止。减操作的次数即为商，剩下的数为余数。

阅读已经给出的 CASL 程序，我们能够看到，程序的最后定义了以 SNO 开始的 5 个数字，即 10000、1000、100、10、1。由此我们可以判断，程序中就是利用这 5 个数来分别求得十进制数的 5 位数字字符的，即将原数除以 100…….就一直十进制数的最后的一位。

5 个需要解答的问题将程序分解得支离破碎，无法从整体上进行把握，只好根据上面对程序算法的推断对程序的局部进行分析，从推断局部的功能入手来解答问题。阅读程序，发现填空（1）后是一个跳转语句，如果不跳转的话，从以后的语句看是把翻译结果从左边开始置"0"，然后 GR2 和 GR1 都加 1。这样我们就能够明确填空（1）的功能，（1）是考查待翻译的二进制整数翻译成十进制数后是否满足 5（4、3、2 等）位，所以应该是 GR0 与以 SNO 为首址的 5 个整数的比较。如何控制呢？是否可以采用"CPL　GR0　SNO　GR1"来完成任务呢？在这个循环中只有 GR1 可以担当这个任务。由此推断在此后的程序中，GR1 一直担当指示 SNO 偏移位置的指针，而且还能够推断 GR3 存放商，GR4 的作用好象是与 GR1 配合，记录已经翻译的倍数。

填空（2）和（3）的连续出现让我们摸不着头脑，但考查（3）和（4）之间的语句，不难尽责这是循环减的语句段，所以（4）应该是一个跳转语句。由于 GR3 的功能所限，不能用 GR3 来控制一个条件跳转，只能构造一个无条件跳转，但跳转的结果是 L2 还是 L3，现在还不能明确。分析这个循环的功能，不难发现，控制循环结束的语句（3）和其后续语句。那（3）应该是一个比较语句，因为如果用减法将影响此后的递减，考虑到处理对象是非负整数，选用 CPL 来构造解答，结合以上对寄存器使用情况的理解，不难构造出（3）的解答为"CPL　GR0，SNO，GR1"，同时还可以确定填空（4）跳转的目标应该是 L3。填空（2）先放在一边。

L4 直至"JNZ　L2"完成向内存中写入一位翻译结果的任务，结合解答（1）时对 GR4 作用的理解，大致可以判定 GR4 的作用是记录已经翻译的位数。依照上面的语句，在这里填写"LEA　GR4，－5，GR1"。

填写完了 4 个答案以后，通读程序，填空（2）的作用和答案就非常明显了，在每一次循环减操作之前一定要给 GR3 清 0，否则商的叠加会使翻译结果莫名其妙，所以在这里填写"LEA　GR3,48"。

[答案]

（1） CPL　CR0，SN0，G,1

（2） LEA　GR 3,48

（3） CPL　GR0，SN0，GR1

（4） JMP　L3

（5） LEA　GR4，－5，G,

试题 5（1996 年试题 4）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言，阅读下列程序说明和 CASL 程序，将应填入（n）处的字句，写在答卷的对应栏内。

[程序说明]

子程序 OFFSET 用二分法，查找无符号整数 M 在一个长度为 N 的有序（升序）无符号整数列表 NTABLE 中的位置。

程序中标号为 LOW 和 UP 的两个存储字分别用作存放查找区间的上下限。

进入子程序时，在 GR1 中给出存放子程序所需参数的起始地址。参数的次序如下：

(GR1)+0	M
1	N
2	NTABLER 的首址

从子程序返回时，GR0 中存放查找结果，即 M 在此有序表中的位置序数。如果表中找不到 M，则 GR0 返回 0，其他寄存器的内容保持不变。

[程序]

```
        START
OFFSET  PUSH 0, GR2
        PUSH 0, GR3
        LD GR0, 0, GR1
        LEA GR2, 0
        ST GR2, LOW
        (1)
        (2)
        ST GR2, P
LOOP    ADD GR2, LOW
```

```
        SRL GR2, 1
        LEA GR3, 0, GR2
        (3)
        (4)
        JZE FOUND
        JPZ INCLOW
        LEA GR2, -1, GR2
        ST GR2, UP
        JMP CMPLU
INCLOW  LEA GR2, 1, GR2
        ST GR2, LOW
        (5)
CMPLU   CPL GR2, LOW
        (6)
        (7)
FOUND   LEA GR0, 1, GR2
        POP GR3
        POP GR2
        RET
LOW     DS 1
UP      DS 1
        END
```

[解析]

试题说明没有提供更多的信息，在这里我们只能了解程序中一些变量的含义。所以问题只能通过两点来解答，一是对程序结构的理解，一是对二分法的掌握程序。二分法的查找过程为：先将待查记录所在的范围（区间）分成两个区间，然后确定待查的记录所在的区间。如此循环往复，直到查找成功或失败。这样，无论如何实现二分法，有些工作是必须的，例如确定查找的上限和下限，确认待查询记录所在的区间。

考察试题所提供的程序，可以推断出程序的对二分法的具体实现过程：首选设指针 LOW 和 UP 分别指向 NTABLE 的第一个元素和最后一个元素，比较 M 与表的第[（LOW＋UP）/2]个元素（[ ]）表示取整），即与区间的蹭元素相比较。若相等，则找到；若 M 大，则说明要找的元素在区间的上半部，所以下一次比较将 LOW 值置为[（LOW＋UP）/2]＋1，UP 值不变；若 M 小，则说明要找的元素在区间的下半部，所以下一次比较将 LUP 值置[（LOW＋UP）/2]－1，LOW 值不变。循环的结束条件是 LOW＞UP，此时查找失败。

结合算法来阅读程序。程序在完成了一些例行工作之后，语句"LD GR0，0，GR1"将要查找的值 M 放入 GR0 中，接下来是设置查找的下限 LOW，显然下面该初始化指针 UP 了。由以上 LOW 的初值为 0，可以推断 UP 的正确初值应该是 N-1。从填空（2）的下一行"ST GR2，UP"来看，UP 的初值是经 GR2 传入的，而且填空（1）、

（2）完成对 UP 的赋初值操作，这样填空（1）、（2）应设为 GR2 等于 N-1，所以（1）、（2）两行应为：

LD GR2，1，GR1；把 N 放入 GR2 中

LEA GR2，-1，GR2；把 N-1 放入 GR2 中

接下来就开始二分法查找，首先求得 UP+LOW 的值，进而求得[（UP+LOW）/2]的值。然后将算出的中间位置值赋给 GR3。显然这里的中间位置只是相对于 NTABLE 的首址而言的。如果在程序中真正用来与 M 进行比较，必须换算成真正的逻辑地址，所以填空（3）的答案为"ADD GR3，2，GR1"，将 NTABLE[（UP+LOW）/2]所在地址赋给 GR3。从接下来的几个跳转语句来看填空（4）的功能应该是进行 M 与 NTABLE[（UP+LOW）/2]比较，然后才有此后根据比较结果进行的跳转。观察"JPZ INCLOW"我们知道，应将 M 值在前，NTABLE[（UP+LOW）/2]在后比较，所以（4）的答案应为"CPL GR0，0，GR3"。

若比较结果相等，程序跳转至 FOUND，这时将位置信息存入 GR0 返回。若 M 大，则跳转至 INCLOW，将 NTABLE[（UP+LOW）/2]+1 的值赋给 LOW。因为 LOOP 循环开始时已假定 GR2 中为 UP 值，所以填空（5）的答案为"LD GR2，UP"，用来恢复 GR2 的值，使 GR2 的值位 UP 的值。若 M 小，则将 NTABLE[（UP+LOW）/2]-1 的值赋给 UP，然后应当判断循环是否结束，即判断 UP 是否小于 LOW，程序中 CMPLU 开始的几句完成这一判断。在 UP>=LOW 时，继续查找，否则应该返回，所以填空（6）的答案应为"JPZ LOOP"。当 UP>=LOW 时，继续查找，否则 UP<LOW，循环结束，返回值应当为 0，以示查找失败。考虑到填空（7）下面的一条语句是将 GR2 加 1 后传送到 GR0，所以（7）的答案应该是"LEA GR2，-1"，与下一条语句结合来完成设置查找失败标志的功能。

[答案]

（1） LD GR2,1，GR1 或 ADD GR2,1，GR1

（2） LEA GR2，－1，GR2

注：（1）、（2）两条指令在 GR2 中形成 N－1 的解答均正确，如：

（1） LD GR3,1，GR1

（2） LEA GR2，－1，GR3

或

（1） LEA GR2，－1

（2） ADD GR2,1，GR1

若在 GR2 中碰 N 可以给 1 分

（3） ADD GR3,2，GR1

（4） CPL GR0,0， GR3

（5） LD GR2，UP

（6） JPZ LOOP

（7） LZA GR2，－1 或 SUB GR2，LOW

注：若（6）、（7）的解答为：

（6） JMI FOUND

（7） JPZ LOOP 或 JMP LOOP 亦可给 2 分

试题 6 （1995 年试题 6）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读下列程序说明和 CASL 程序，把应填入（n）处的字句，写在答卷的对应栏内。

[程序说明]

本程序是按某种规律自动生成 16＊16 单色点阵图形。点阵图形中每行 16 点的信息排列在一个存储字中，16＊16 点阵图形可以用连续的的 16 个存储字来表示。

程序中标号为 PTN 开始的 16 个存储字连续存放自动生成 16＊16 点阵图形，点阵图形中的第一行作为已知数据给定，其余 15 行按下述规则自动生成：

1． 每个存储字的第 0 位和第 15 位（即边界点）恒为 0。

2． 一个存储字的第 n 值 0＜n＜15 取决于前一存储字的第（n-1）位和第(n+1)位的值是否相同。若这两位的值相同，则第 n 位为 0，否则为 1。

3． 例如，点阵图形第一行的存储字内容为＃35B4，按上述规则生成点阵图形第二行的存储字内容为＃71B2，其余类推。

0011010110110100 #35B4
0111000110110010 #71B2
0…… 0 ……..

[程序]

```
        START
        LEA GR1, 0
        LD GR0, PTN, GR1
LOOP    SLL GR0, 2
        (1)
        (2)
        (3)
        LEA (4)
        (5)
        LEA GR2, -15, GR1
        JMI LOOP
        EXIT
PIN     DC #35B4
        DS 15
        END
```

[解析]

粗读程序，应该发现需要解答的问题非常集中。研究程序结构，不难确定需要解答的部分大致覆盖了程序的核心部分，生成并存储各位。这样，解答的关键就是如何从有限的语句中推断程序所使用的算法。

从试题说明中不难看出，点阵的每个存储字的第 0 位和第 15 位为 0。除了第 1 个存储字外，每个存储字的第 n 位取决于上一个存储字的第 n-1 位的和 n＋1 位的值是否相同，也即由上一个存储字的第 n－1 位的和第 n＋1 位异或而得。

阅读程序，可以推断出在标号 LOOP 和语句"JMI　LOOP"之间的语句完成生成一个存储字的操作，因为作为最外层的循环其循环体执行了 15 次。是否在需要解答的部分还有一个循环需要构造？从语句数量上来看是不够的，因为 5 条语句除了循环变量的初始化、循环变量的修改和跳转之外剩余的两条语句已经难以完成取两位异或并存储的任务，看来程序采用的是另一种算法。

考察语句"LD　GR0，PTN，GR1"和语句"SLL　GR0,2"的功能。由于需要生成的存储字第 0 位和第 15 位的值是固定的，我们需要解决的是中间的 14 个二进制位的值，把当前存储字左移 2 位并与当前存储字异或应该是比较简便的方法。我们姑且认定程序采用的就是这种算法。

如果采用上述算法，现实的问题就是在新生成的存储字中，第 0 位到第 13 位是符合要求的存储字的第 1 位到第 14 位，而最后两位的值是不确定的。为达到第 0 位和第 15 位为 0 的目的，我们可以采用逻辑移位的方法，可以采用的方法是先逻辑左移 2 位，再逻辑右移 1 位。把上述算法用 CASL 语句表达，可以填写在填空（1）、（2）和（3）处。仔细推敲答案，上述语句在功能实现上没有问题，这也印证了我们上面的推断。

由于在填空（5）之后有通过对 GR1 减 15 以确定是否结束循环的语句，在（4）和（5）中应该有一个对 GR1 进行操作的语句，否则不但循环无法结束，就是以后的循环也难以进行正确的 LD 操作；在上面已经生成了一个符合要求的存储字，应该将其存储进 PTN 之后的连续存储空间内，这项任务也应该在填空（4）和（5）中完成。老鸦到（4）所在的语句是一个 LEA 操作，我们在（4）中填写"GR1,1GR1"以完成第 1 项任务，在（5）中填写"ST GR0，PTN，GR1"以完成第 2 项任务。

在所有需要解答的问题都解答完毕之后，我们再从整体上阅读程序，以确定解答的正确性。本题给我们的启示就是在无法看出程序所使用的算法时，应该通过各种迹象来推断程序所使用的算法，以这个推断为出发点解答题目。推断算法的关键就是对程序中问题前后的语句的功能进行充分的理解，交考察它们对解答题目的直接意义。

［答案］

(1) EOR GR0,PTN,GR1

(2) SRL GR0,2

(3) SLL GR0,1

(4) GR1,1,GR1

(5) ST GR0,PTN.GR1

试题 7（1994 年试题 6）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读程序说明和 CASL 程序，把应填入（n）处的字句，写在答卷的对应栏内。

［程序说明］

子程序 DIVIDE 将真分数 a/b(0<a<b)转换成 N 位小数(0<N≤20)，并输出结果。

子程序从栈中接受参数。参数的入栈顺序为：分子（a）、分母（b）和小数位数（N）。进入子程序时，已保证 0<a<b 且 0<N≤20 子程序不再对参数作合法性检查。

由于需要将结果输出，所以子程序中计算出每一位小数后直接用字符方式顺次存放在标号 OUTPUT 之后的存储区域中。数字字符 0~9 的 ASCII 编码依次是 48~57。

[程序]

```
        START DIVIDE
DIVIDE  LEA GR3, 2
        LD GR1, 1, GR4
        LEA GR1,2 GR1
        ST GR1, OUTLEN
        LD GR0, (1)
DIGIT   LEA GR2, 48
        SLA GR1, 1
        ST GR0, WORK
        SLA GR0, 2
        ADD GR0,WORK
SUBST   CPA GR0, (2)
        JMI GOOD
        (3)
        (4)
        JMP SUBST
GOOD    ST GR2, OUTPUT, GR3
        LEA GR3, 1 GR3
        SLL (5)
        JZE OUT
        CPA GR3, OUTLEN
        JMI DIGIT
OUT     ST GR3,OUTLEN
        OUT OUTPUT, OUTLEN
        RET
OUTPUT  DC '0'
        DS '20'
OUTLEN  DS 1
WORK    DS 1
        END
```

[解析]

阅读试题说明，首先确定子程序从栈中接收参数时各个参数的地址，我们能够看出，a、b、N 的地址分别是 GR4＋3、GR4＋2、GR4＋1。

仔细研究程序，发现 N 加 2 之后赋给 OUTLEN，这是因为输出时需要输出"0"。填空（1）所在的语句究竟要给 GR0 中装入什么内容呢？应该是 GR4＋3 或 GR4＋2。继续阅读子程序，发现 GR0 的内容在以下的语句中被乘以 10，从一般的编程常识来看，在求真分数　时被乘以 10 的大多是分子，当然，这仅仅是常识而已，至于是否在本题中适用，还要继续推敲才能够确定。我们暂且认为填空（1）的解答是"3，GR4"，看下面的程序是否能够证明这种推断的正确性。

填空（2）究竟将 GR0 与什么进行比较？从后续语句来看，如果 GR0 的值小于被比较的值，就跳转互标号 GOOD。仔细阅读标号 GOOD 及其以后的语句，不难发现这是存储一个小数位交判断是否结束子程序进行输出的语句段。那么（2）中被比较的内容应该是分母（GR4＋2），所以填空（2）的解答就是"2，GR4"。

从程序结构来看，（3）和（4）应该完成求得一位小数的操作。CASL 中没有除法指令的，一般采用减法来实现除法操作。从后面的语句"格 ST　GR2，OUTPUT　，GR3"来看，试题中求得的小数位存储在 GR2 中，这样我们就不难构造（3）和（4）中的解答来完成除法操作了，即"SUB　GR0，　2，　GR4"和"LEA　GR2,1，GR2"。填空（5）所在的语句是判断当某个值为 0 是结束子程序，那么这个值是什么呢？显然是当分子为 0 时应该结束子程序，所以应当填写"GR0,0"。

［答案］

（1）　3，GR4

（2）　2，GR4

（3）　SUB　GR0,2，GR4

（4）　LEA　GR2，1，GR2

（5）　GR0,0

试题 8（1993 年试题 6）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言，阅读程序说明和 CASL 程序，把应填入（n）处的字句，写在答卷的对应栏内。

［程序说明］

子程序 MOVE 是将地址为 A 开始的 N 个存储单元移动到地址为 B 开始的 N 个存储单元中，对于两个区域互相重叠的情况也能正确处理。

主程序在 GR1 中给出存放子程序所需参数的起始地址，参数的存放形式为：

(GR1)+	A
0	B
+1	N
+2	

［程序］

```
MOVE    START
```

```
        LD GR2,1, GR1
        LD GR3, 0, GR1
        CPL GR3, 1, GR1
        JZE ENDMOV
        (1)
        (2)
        JMP SAVE
LT      (3)
        LEA GR3, (4)
        (5)
        LEA GR3, (6)
        (7)
SAVE    ST GR0, WORK
        LD GR1, 2, GR1
LOOP    LD GR0, 0, GR3
        ST GR0, 0, GR2
        ADD GR2, WORK
        ADD GR3, WORK
        LEA GR1, -1, GR1
        JNZ LOOP
ENDMOV RET
WORK    DS 1
        END
```

[解析]

细阅读程序说明，注意程序说明中对两个区域重合情况的特别强调，注意 GR1 中给出的是子程序所需参数的地址，而非参数本身，理解了这一点对整个子程序的理解非常重要。

从子程序的开头部分阅读，"START"后面 4 条语句的作用是易于理解的，前前句把 B 的起始地址装入 GR2，把 A 的起始地址装入 GR3，第 3 句比较 A、B 起始地址的大小，第 4 句处理 A、B 起始地址重合的情况，在这种情况下可以不必移动。我们可以先不考虑接下来的填空,先对程序的整体结构进行分析。很明显,在"JZE ENDMOVE"之后到标号 SAVE 之前没有发生移动存储单元的动作,因为"LD GR1,2,GR1"到"JNZ LOOP"之间完成了一次 N 个存储单元的传送过程，由此我们推断

本题的 7 个需要填写都集中在处理 A、B 地址不同时的传递控制参数设置上。要处理好 A、B 地址重叠的情况，必须考虑两种情况，即当 A 大于 B 时，作顺序传送，这样即使 B＋N＞A，也能保证传送处理的正确性；当 B 大于 A 时，作反序传送，这样即使 A＋N＞B，也能保证传送处理的正确性。

我们先看填空（1）和（2）之后的"JMP SAVE"，考查在直接跳转到标号 SAVE 时的操作。跳转后的第一条语句是"ST GR0, WORK"，本子程序中尚无对 GR0 操作的语句，如果此前没有对 GR0 的操作，程序的结果将是不可预知的。要保证

程序的正常运行，在"JMP SAVE"之前必须有对 GR0 进行操作的语句，现在的问题是在什么位置进行何种操作。继续阅读子程序，易知"JMP SAVE"发生在 A 大于 B 的情况下。标号 SAVE 后的语句完成从 A 到 B 的搬运，" WORK"（GR0）是作为传送地址移动而准备的。由此推断从标号 LT 到标号 SAVE 之间的语句是为反序传送进行传送参数设置。

有了以上的分析，（1）和（2）的答案就是显而易见的了，填空（1）应该是在 A 小于 B 时的跳转，跳转的目标应该是"LT"。而填空（2）的作用就应该是对 GR0 的操作，从"ST GR0"、"WORK"及"WORK"作为地址修改参数两点来看，这里应该是"LEA GR0,1"，以保证顺序传送的进行。

既然标号 LT 和标号"SAVE"之间是为反序传送设置传送参数，反序传送时传送的源数据区是从 A＋N－1 开始（注意存储区的地址计数是 1 从 0 开始的），而传送的目的数据区则是从 B＋N－1 开始，地址修改参数应该是－1。结合以上分析，在程序执行到标号 SAVE 之前时，GR3 的内容应该是 A＋N－1，GR2 的内容应该是 B＋N－1，GR0 的内容应该是－1。在填空（3）到填空（7）之间完成上述操作解决方案是灵活的，我们可以在（3）或（7）处填写"LEA GR0，－1"以完成对 GR0 的设置，但考虑程序的可读性，将"LEA GR0， －1"填写在（7）处比较好些。蓁其余的 4 条语句就是完成对 GR2 和 GR3 进行设置的 A＋N－1 和 B＋N－1，结合程序说明，完成该操作的语句应该很明确了。

本题解答关键在于理解填空（1）到（7）之间语句的作用和反序传送控制参数的主。设置，如有一方面的内容不能正确理解，就不能得出正确的解答。本题需要解答的部分比较集中，关键是我们要结合程序的整体结构，确定需要填写的部分所承担的功能，从实现程序功能入手，确定需要填写的语句。如果不结合程序整体结构进行考虑，面对填空多于语句的试题，我们就会无所适从。

[答案]

（1） JMI LT

（2） LEA GR0,1

（3） ADD GR3,2 GR1

（4） －1 GR3

（5） ADD GR2,2，GR1

（6） －1 GR2

（7） LEA GR0，－1

试题 9（1992 年试题 6）

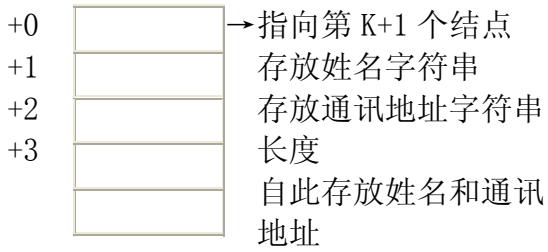
在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言，阅读程序说明和程序，回答问题，把解答写在答卷的对应栏内。

［程序说明］

本程序根据输入的姓名（字符串），在已有的线性璷中查找其相应的通讯地址（字符串），并予以输出。

标号为 DATPTR 的存储字存放着线性表第一个结点的地址。结点的数据结构定义如下：

第 K 个结点



最后一个结点的指针字段为空指针（内容为 0）。

［程序］

```
        START GEGIN
OTLONG  DS 1
OTTEXT  DS 80
NOLONG  DC 10
NNOTEXT DC 'NOT FOUND'
INTEXT  DS 80
INLONG  DS 1
DATPTR  DC FIRST
BEGIN   IN INTEXT, INLONG
        LEA GR0,0
        CPA GR0, INLONG
        JZE ENDSCH
        LEA GR3, DATPTR
NEXTMEM (1)
        (2)
        JZE NOTFND
        LD GR0, 1, GR3
        CPA GR0,INLONG
```

```
        JNZ NXMEM
        (3)
        LEA GR2, INTEXT
        CALL CMPSTR
        (4)
        LD GR0, 2, GR3
        ST GR0, OTLONG
        (5)
        (6)
        LEA GR2, OTTEXT
        CALL CPYSTR
        OUT OTTEXT,
        OTLONG
NOTFND  JMP BEGIN
        OUT NOTEXT,
ENDSCH  ONLONG
CMPSTR  JMP BEGIN
CMPNXT  EXIT
        PUSH 0, GR3
        LD GR3, 0, GR1
        CPL GR3, 0, GR2
        JNZ CMPEND
        LEA GR1, 1, GR1
        LEA GR2, 1, GR2
CMPEND  SUB GR0, CONST1
        JNZ CMPNXT
CONST1  POP GR3
CPYSTR  RET
CPYNXT  DC 1
        PUSH 0, GR3
        LD GR3, 0, GR1
        ST GR3, 0,GR2
        LEA GR1,1,GR1
        LEA GR2, 1,GR2
        SUB GR0, ONE
        JNZ CPYNXT
        POP GR3
        RET
        ONE DC 1
        END
```

[解析]

钥匙首先要弄清楚程序中所使用的数据结构，这是理解程序处理流程的基础。由程序说明我们应该能够发现该程序使用的是链表。链表的表头由 DATPTR 指示，每个结点存储 5 部分信息，即第 K＋1 个结点的存储地址、姓名字符串长度、通信地址字符串长度、N1 个存储字存放姓名、N2 个存储字存放通信地址，而最后一个结点的第一个字节的内容是 0，这是链表未尾的标志。程序中进行的比较查找就是以这个链表为基础的，理解了链表结点的结构，理解程序的处理流程就会比较容易了。然后再对程序的整体结构进行考查，发现程序由主程序和两个子程序组成，我们要解答的问题都在主程序内。从子程序的名字上大致可以判断它们的功能，仔细考查，确认 CMPSTR 完成字符串的比较，CPYSTR 完成字符串的拷贝，一定要雇它们都使用哪些寄存器工作，各寄存器的作用是什么。

阅读程序，在明显的 4 组变量定义之后，程序开始进行输入操作及输入后简单的输入内容检查，这是一些无关大局的细节问题。接下来的"LEA　GR3，DATPTR"让我们得知 GR3 的内容就是链表表头的地址。

标号 NXTMEM 后面的两填空让人感到题目的难度，但结合变量的含义仔细推敲，"JZE　NOTFND"的含义是，如果上个语句的结果使 FR 值为 00，则宣布没有查找到符合要求的结点。查找链表则只能在查找到链表未尾时才可以宣布结束，而在这里链表结束的标志是结点的第一个字节内容为 0。经考查，断定此处两空白应该完成装入结点首字节和判断结点首字节的任务，但目前几乎每个寄存器都是可用的，究竟使用哪个寄存器，我们暂且放下，继续阅读程序。后续的语句是比较输入字符串的长度与该结点的长度是否相等，此处使用 1，GR3 来表示姓名长度，则易知上面的操作应该使用 GR3。这里需要注意的是语句"GR3，SATPTR"有稳操胜券不敢轻易使用 GR3，但结合后续语句可以判定不使用 GR3 就难以实现结点指针的移动。

在姓名长度相等的情况下，应该调用 CMPSTR 比较咀输入字符串相等，很明显，CMPSTR 需要用 GR0、GR1 和 GR2 传递参数，其中 GR0 传递需比较的字符串的长度，GR1 和 GR2 分别传递需要比较的字符串的地址。GR0 是无须再设置的，GR2 的内容由"LEA　GR2，INTEXT"来设置，且明确了 GR2 传递的是输入字符串的地址，那么填空（3）处就应该是设置 GR1 的内容为结点姓名字符串的地址，答案是明显的。CMPSTR 的调用后面紧跟着一个填空，我们可以暂且忽略它，继续阅读下面的程序。下面的程序先是调用 CPYSTR，然后双输出字符串，这是找到符合条件的结点之后的工作。由此我们判断，填空处理 CMPSTR 认定当前结点并非查找结果的情况，应该是跳转到 NXTMEM，但如何表示呢？考察子程序 CMPSTR，发现跳转受"SUB　GR0，CONST1"控制，因为若认定当前结点符合条件，则一定是比较到字符串的末尾，"SUB　GR0，　CONST1"应该使 FR 的值为"00"，则 FR 值不为"00"时发生跳转，所以此处应该填写"JZN　NXTMEM"。

填空（5）和（6）后面是对子程序 CPYSTR 的调用，结合我们对 CPYSTR 的理解，易知该子程序使用 GR0、GR1 和 GR2 传递参数，GR－传递需拷贝的字符串的长度，GR1 拷贝对象的地址，GR2 传递拷贝目的字符串地址。很明显这里要拷贝的应该是通信地址，GR0 和 GR2 的设置已经完成，需要把通信地址字符串的地址放进 GR1，考查链表结点的结构，不难发现通信地址字符串的确切地址必须由姓名地址加姓名长度求得，所以就很难用一个语句来完成工作，并且此处出没有其他得的工作

要做，所以我们在这里填写"LEA GR1， 3， GR3"和"ADD GR1,1，GR3"， 结点的通信地址字符串首地址送到GR1。

由以上的分析解答可以总结出两点可以称作经验的东西：1、要切实理解试题说明；2、要切实理解程序流程和结构。

［答案］

(1) LD GR3, 0, GR3

(2) LEA GR3, 0, GR3

(3) LEA GR1, 3, GR3

(4) JNZ NXTMEM

(5) LEA GR1,3,GR3

(6) ADD GR1,1,GR3

试题 10（1991 年试题 6）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言，阅读程序说明和程序，回答问题，把解答写在答卷的对应栏内。

［程序说明］

本程序将存储字 A 中的第 n 位起的 n 位二进制位搬至存储字 B 的第 b 位起的 n 位中。

1、 主程序在 GR1 中给出子程序所需的参数起始地址。

2、 子程序所需的参数个数和含义如图 7－1 所示。

3、 0≤a≤15; 0≤b≤15;1≤n≤16; a+n≤16;b+n≤16。

4、 存储字 B 中，除中开始的 n 位外的其他二进位应保持原值。

［问题］

在程序中（1）～（8）处各填入一条正确的指令，以完成此程序。除非必要，标号栏还要填写。

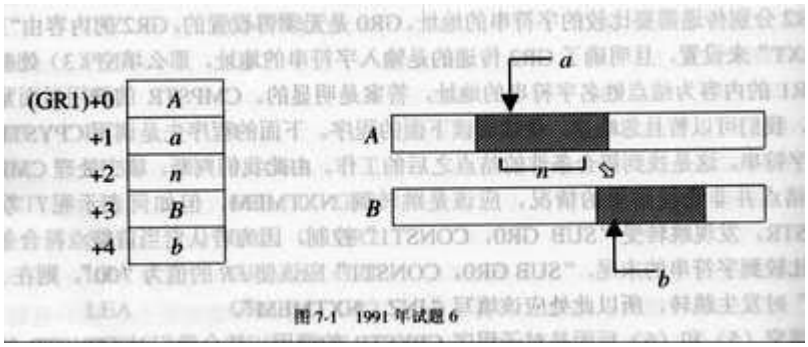


图 7-1 1991 年试题 6

[程序]

```
MOVBIT  START
        LD GR2, 2, GR1
         (1)              SAVE1 中形成
         (2)
        ST GR2,    SAVE1  11…   00……0
        LD GR2,1, GR1        ↓
         (3)                n个
        LD GR2,0, GR1
        AND GR0,0, GR2
        LD GR2,4, GR1
         (4)
        LD GR2,1, GR1
        SUB GR2,4, GR1
         (5)
        JMP SHFTE
SHFTR   SUB GR2,1, GR1
         (6)
SHFTE   ST GR0, SAVE2
        LD GR2, SAVE1
        LD GR3,4, GR1
         (7)
         (8)
        LD GR3,3, GR1
        AND GR2,0, GR3
        OR GR2, SAVE2
        ST GR2,0, GR3
        RET
C8000   DC #8000
CFFF    DC #FFFF
SAVE1   DS 1
SAVE2   DS 1
        END
```

［解析］

仔细阅读程序说明，把握程序要实现的功能和试题所提供的信息。重点是要理解GR1中给出了子程序所需参数的起始地址，并从GR0开始的各内存单元所存储的内容。然后考查子程序的变量定义部分，根据变量值的特点估计它们的用途。

子程序的开始部分需要填写的内容比较多，虽然有一句注释，但对我们理解程序帮助不大。在这种情况下我们应该跳过这些语句，直接阅读填空（3）之后的部分，看看（3）之后进行了什么操作，则易知在（3）之前已经进行了哪些操作。从"LD　GR2，0，GR1"读起，推断到该句和"AND　GR0,0，GR2"两句，程序应该是已经完成了从A中取出从a位起的n位，因为在此之后的工作已经是比较a和b的大小　，这是进行对准数位的前奏，一旦对准数位，搬运工作就开始了。这样，到填空（3）处，GR0中的内容应该具有这样的特点，即从该字的第a位起，有连续n个1，其余的数位全部是0。考察已有的3条语句，没有完成此项功能的相应语句，不过"LD　GR2,2，GR1"和"LD　GR2,1，GR1"两条语句也能给我们许多有用的信息，要想在GR0中形成符合要求的内容，必须先生成一个在开头或结尾具有n(GR1+2)位1、其余位全部为0的字，然后再将其逻辑左移或右移若干位(由GR1＋1决定）。综合考虑到填空（4）之前没有其他要完成的任务，我们推断（1）至（3）完成上述操作，即在GR0中形成从第a位起有连续n位的1的存储字。结合对变量的，我们在3处分别填写"LD　GR0，C8000"、"SRA　GR0，－1，GR2"和"SRL　GR0,0，GR2"。

跳过"CPA　GR2,1，GR1"后面的空白，继续阅读后面的语句，不难确定到标号SHFTE之前完成的是对位工作，即将从A中取出的从第a位起有连续n个1的字变成从第b位起有连续n个1的字。很明显，填空（4）之后直到"JMP　SHFTE"完成的是a大于b的对位工作，而标号SHFTE及后续的一句完成a小于b的对位工作。两条SUB语句已经在GR2中生成了需要移动GR0的内容的位数。当a大于b时，GR0内容左移，否则右移，填空（5）和（6）的答案非常明显。现在回头去考察（4），（4）的跳转功能已经非常突出，即在b大于a（a小于b）时跳转到标号SHFTR所在语句。

要想把GR0的内容搬运到B中，还必须将B从第b位起的连续n位都置0，否则后面的"OR　GR2，SAVE2"就难以完成任务，结合"OR　GR2，SAVE2"所提供的信息，可知将B从第b位起的连续b位都置0的字应该生成在GR2中。由此上溯，易知填空（7）和（8）完成的是在GR2中形成的除第b位起的b位为0外，其余数位全部为1的字。综合考虑用过的和未用过的变量及应该填写的语句数量同，此处的答案为"SRL　GR2,0，GR3"和"EOR　GR2，CFFF"。

总结解析该题的经验，当我们无法按语句顺序理解程序时，可以先阅读后面的语句，确定不易理解的部分应该完成哪些任务，然后结合实际情况，综合考虑程序的整体，排队错误判断之后解答问题，结合上下文在规定的位置填写语句，完成

后续工作需要完成而又没有完成的任务。本题中的（1）、（2）、（3）与（7）、（8）的解答过程就是这样的，我们可以称其为"倒卷珠帘"。

［答案］

(1)LD GR0, C8000

(2)SRA GR0，－1，GR2

(3)SRL GR0,0，GR2

(4)JPZ SHFTR

(5)SLL GR0, 0, GR2

(6)SRL GR0, 0, GR2

(7)SRL GR2, 0, GR3

(8)EOR GR2, CFFF

试题 11（1990 年试题 7）

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言，阅读程序说明和程序，回答问题，把解答写在答卷的对应栏内。

［程序说明］

本程序完成两个 4 位十进制数相加，并输出与会代表两数之和。

例：输入"5794＋6438"输出"12232"。

1、 按上述要求输入，否则输出"INPUT ERROR"，信息，并重新输入。

2、 从低位开始，逐位进行十进制数相加。

3、 若输入长度为 0 时，本程序结束。

［问题］
在程序中（1）～（7）处逐个填入一条正确指令，以完成此程序。除非必要，标号栏还要填写。

［程序］

```
标号    指令码操作数
        START
BEGIN   ST      GR4, SPW
RETRY   IN1     INBUF, LENG
        LEA     GR1,0
        CPA     GR1, LENG1
        JZE     HALT
        LEA     GR2, 9
        CPA     GR2, LENG1
        JNZ     ERROR
        LEA     GR3, 4
        (1)
        CPL     GR0, SING
        JZE     PASS1
ERROR   OUT     INERR, LENG2
        JMP     RETRY
PASS1   LD      GR1, SM
        ST      GR1, INBUF, GR3
        LEA     GR1, 0
LOOP1   LD      GR2, INBUF, GR1
        LEA     GR1,1,GR1
        CPL     GR3, SM
        JM1     ERROR
        CPA     GR3,LM
        JPZ     ERROR
        AND     GR3, BCD
        PUSH    0, GR3
        (2)
        JNZ     LOOP1
        ST      GR2, CY
        LEA     GR1,4
        (3)
        ADD     GR3, (4)
        ADD     GR3, CY
        CPA     GR3, TEN
        JMI     LAB1
        ADD     GR3,SIX
        ADD     GR3,BCD
        (5)
        JMP     LAB2
LAB1    LEA     GR0,0
LAB2    ST      GR0,CY
        OR      GR3, SM
        ST      GR3,OUTBUF,GR1
```

```
        (6)
        JNZ     LOOP2
        OR      GR0,SM
        (7)
        OUT     OUTBUF, LENG3
        JMP     RETRY
HALT    LD      GR4,SPW
        EXIT
SPW     DS      1
INBUF   DS      80
LENG1   DS      1
INERR   DC      'INPUT ERROR'
LENG2   DC      11
SING    DC      '+"
SM      DC      '0'
LM      DC      ':'
BCD     DC      #000F
CY      DS      1
TEN     DC      10
SIX     DC      6
OUTBUF  DS      5
LENG3   DC      5
        END
```

[解析]

首先阅读程序说明，程序说明提供的有效信息不多；然后阅读程序后半部分的变量定义，切实各变量的含义。粗读程序，发现必须在某处标号栏填写标号 LOOP2。从 GEGIN 开始读起，很快就会发现一直到"JPZ　RETRY"是关于输入合法性检查的部分。"LEA　GR3,4"后面是一个我们必须填写的填写，我们暂且跳过两行，看填后面的语句"CPL　GR0，SING"很明显，在检测输入的第 5 个字符是否为"＋"，而且该字符已经被装入 GR0。仔细推敲，可以确定将输入的第 5 个字符装入 GR0 的操作只能是由"CPL　GR0，SING"前面的两行语句完成，因此，"LEA　GR3,4"的作用也就很明显了，这是在为取内存的内容作地址偏移准备。至此，填空（1）的答案也就可以确定了。

继续阅读程序，不难推断从"LEA　GR1,0"到"JNZ　LOOP1"构成一个循环，在跳转之前修改循环计数器属于常识，这里重要的是判断哪个寄存器或变量循环是计数器，循环步长是多少。对程序进行分析，发现该循环仍旧是进行输入合法性检查的，它检查输入缓冲区的每一位是否为"0"－"9"，并将其转换为数字，按输入顺序入栈。应该注意到，标号 PASS1 处已经把输入"＋"变成"0"，所以这里的合法性检查能够顺利进行。既然是对输入缓冲区的每一位进行检查，步长只能为 1；从"JNZ　LOOP1"来看，循环计数器应该是递减的。结合以上对程序的理解，这个

递减的变量必须是合法的输入长度，这个长度现在存储在 GR2 中，所以这条修改循环条件的语句就应该是"LEA GR2，－1，GR2"。

在上面的循环完成以后，两个 4 位数已经按输入顺序入栈，接下来程序的最后一部分应该是进行两数相加。从"LEA GR1,4"到"JNZ LOOP2"又是一个循环。但 LOOP2 这个标号到底放在哪儿呢？填空（3）、（5）、（6）都有可能，但仔细分析，产生排除了放在（6）处的可能性。（5）后面是一个无条件跳转语句，以该语句为循环体的第二条语句是不合情理的。这样，标号 LOOP2 只能放在（3）处。分析从（3）到"ADD GR3，CY"，可以推断它们的任务是完成对应数位的相加。由于前面我们已经把输入内容压栈，而（3）后的语句又有 GR3 参加，我们就可断定（3）是对 GR3 的操作。因为对 GR3 的操作不足以使其承担这样的任务。所以（3）的答案应该是"LOOP2 POP GR3"。后续语句完成数位的相加，如果在（40 中也使用 CY 的话，共同性恐怕就不是几条语句所能完成的任务了。由于上面已经将所有输入压栈，我们可以通过地址偏移来实现对堆栈内容的操作。从前面压栈过程易知，对应的数位在栈中相隔 5 位，似乎应该在（4）填空"5，GR4"，但如果注意到 POP 指令对堆栈指针的修改，我们就会将"5"改为"4"，这是比较容易理解但比较容易忽略的事情。

接下来的进位检查和进位处理，当和大于 10 时，要产生进位，参考"LAB1 LEA GR0,0"和"ST GR0,CY",易知（5）处应该填写"LEA GR0,1".

一直读到"JNZ LOOP2",未发现循环体中修改循环条件的语句，GR2 的值为 0，是不能和 JNZ 相互配合的，GR3 和 GR4 各胡用途，只有 GR1 尚在闲置，而且其值为 4，正符合我们进行的 4 位数相加的要求，通过对其递减和"JNZ LOOP2"的判断，完成循环控制是非常方便的。考察前后语句，未发现与其他工作的冲突，所以我们就确定 GR1 是作为循环计数器的。

在循环 LOOP2 结束后，两个 4 位十进制数的和可能会在最高位产生进位，与上面的操作相同,我们要把它转换成字符形式准备输出,理解最后几条语句的含义后，（7）完成将进位存入 LUTBUF 的功能应该是很明显的，所以这里我们填写"ST GR0，OUTBUF"。

[答案]

（1） LD GR0，INBUF，GR3

（2） LEA GR2，－1，GR2

（3） LOOP2 POP GR3

（4） 4，GR4

（5） LEA GR0,1

（6） LEA GR1，－1，GR1

（7） ST GR0，OUTBUF/

7. C 语言程序编制试题精解

第八章 C 语言程序编制度量精解

高级程序员级 C 语言编程试题有一定的浓度和难度。根据新的考试大纲，每年有两道必考的 C 语言编程试题，考试试题更注重于算法理解和实现方面，在解答试题的过程中，一定要记住要严格按照试题说明提供的算法展开思路，而不是在了解了程序的功能后按照自己推断的算法去理解程序。这一点有必要在复习备考过程中加以重视，具体的方法就是在阅读试题说明过程中不产生自己的算法中，以理解并接受试题说明的内容为目标。

试题 1（2000 年试题 5）

阅读下列程序说明和 C 代码，就填入（n）处的字句在答卷的对应栏内。

［程序说明］

下列文法可用来描述化学分子式的书写规则（例如，AL2（CO3）3、C u （OH）2）：

λ → β | β λ

β → δ | δ n

δ → ξ | ξ θ |（λ）

其中 λ 是一个分子式； δ 或是一个元素，或是一个带括号的（子）分子式；元素或是一个大写字母(记为 ξ),或是一个大写字母和一个小写字母(记为 ξ θ)；β 或是一个 δ，或是在 δ 之后接上一个整数 n； δ n 表示 β 有 n 个 δ 的元素或（子）分子式。一个完整的分子式由若干个 β 组成。

当然一个正确的分子式除符合上述文法规则外，还应满足分子式　本身的语义要求。

下面的程序输入分子式，按上述文法分析分子式，并计算出分子式的分子量。例如：元素 H 的原子量是 1，元素 0 的原子量是 16。输入分子式 H20，计算出它的分子量为 18（1＊2＋16）。程序中各元素的名及它的原子量从文件 atom.dat 中读入。

［程序］

```
#include <stdio.h>
#include <string.h>
#define MAXN 300
#define CMLEN 30
struct elem {char name[3]; /* 元素名 */
Double v; /* 原子量 */
}nTbl [MAXN];
char cmStr [CMLEN], *pos;
int c; FILE *fp;
double factor();
double atom() /* 处理文法符号 */
{ char W[3];int i; double num;
while ((c=*pos++)= =''||c= ='\t'; /* 掠过空白字符 */
if (c= ='\n')
return 0.0;
if(c>='A'&& c<='z') { /* 将元素名存入 W  */
w[i=0]=c; c=*pos++;
If (c>='a'&& c<='Z') w[++i]=c;
Else pos--;
W[++i]='\0';
For (I=0;nTbl[i],v>0.0;i++)
If(strcmp(w,nTbl[i],name)= =0)
return nTbl[i],v;
Prinft("元素表中没有所输入的元素：\t%s\n",w);
return -1.0;
} else if (c= = '(') {
if((num= (1) )<0.0)
return -1.0; /* 包括可能为空的情况 */
if(*pos++!=')')
{ printf ("分子式中括号不匹配！\n"); return -1.0
return num;
}
prinft("分子式中存在非法字符：\t%c\n",c);
return -1.0;
}
double mAtom() /* 处理文法符号 β */
{
double num; int n=1;
if((num= (2) )<0.0)
return -1.0;
c=*pos++;
if(c>='0'&& c<='0')
n=0;
while 9c>='0'&& c<='9')
```

```
{
n= (3) ;
c=*pos++;
}
}
pos--;
return num *n
}
double factor() /* 处理文法符号 λ */
{
double num=0.0,d;
if((num=mATom())<0.0)
return -1.0;
while)*pos>='A'&& pos<='Z'||pos= ='('
{
if((d= (4) )<0.0)
return -1.0;
(5) ;
}
}
void main()
{
char fname[ ]="stom.dat"; /* 元素名及原子量文件*/
int i;double num;
if((fp=fopen(fname,"r"))= =NULL) /* 以读方式打开正文文件 */
{
prinft("Can not open %s file.\n",fname);
return; /* 程序非正常结束 */
}
i=0;
while (i<MAXN && fscanf (fp,"%s%1f",nTb1[i].name,& nTb1[i].v= =2)
i++;
fclose(fp);
nTb1[i].v=-1.0;
while(1)
{ /* 输入分子式和计算分子量循环，直至输入空行结束 */
prinft("\n 输入分子式！（空行结束）\n";gets(cmStr);
pos=cmStr;
if(cmStr[0]= ='\0')
break;
if(num=factor())>0.0)
if(*pos!='\0')
printf("分子式不完整！\n");
else
```

```
printf("分子式的分子量为 f\n",num);
}
}
```

[解析]

首先要仔细阅读并理解程序说明中所提供的文法规则。只有充分掌握这些规则，解答才会有规律可循。在下面的解析中我们将始终结合方法规则来解答试题。

粗读程序，不难发现程序中对 3 个函数的调用都很简单，由于字符串指针 pos 是全局变量，各函数在进行处理时直接对 pos 进行处理，然后返回自己所计算的那一部分分子式的分子量。

对程序的阅读要从主函数 main()开始。从第 51 行到第 64 行都很好理解，一直到第 65 行才能看到对函数 factor()的调用。很显然，在调用函数 factor()后，返回值 num 即是分子式，当然，在分子式不合乎文法规则时， num 是一个负数。那么我们就开始研读函数 factor()。

我们先看关于的文法规则： λ → β \ β λ 。这就是说， λ 或者是由 β 构成，或者是由构成。处理的时候应该区分这两种情况，即处理完了 β 之后应该判断其后是否有 λ，如果有 λ，应对其进行处理。

那么程序中是如何表述的呢？很明显，第 45 行处理了 λ 起始部分的 β 。在没有涉及到函数 mStom() 更多内容时，我们只要知道 mAtom()的返回值（分子量或者是表示分子式有问题的－1.0）赋给 num 就可以了。第 46 行的 while 循环条件如何理解？结合以上对文法规则：、的分析，我们不难判断，这是判断其后是否还有一个 λ 或 β 。第 47 行显然应该是一个函数调用语句（比较第 65 行和 45 行，不难发现本程序中调用函数的常用语句形式）。这里应该调用什么函数？处理 λ 和 β 的过程基本相似，这里可以调用函数 factor()，也可以调用函数 mAtom()。

基于以上的理解，在函数调用成功的情况下，整型变量 d 的值应该是函数调用返回的分子量。从第 49 行我们可以判定，函数 factor()中整型变量 num 是用来存储分子量的。那么第 47 行的函数调用返回值就应该累加到 num 中，这是一个简单的操作，其 C 语言实现也很容易。
在研读函数 mAtom()之前，我们先看看关于 β 的文法规则： β → δ | δ n。这就是说， β 或者由 δ 构成，或者由 δ 和一个整数 n 构成。这样，在处理完了 δ 后，一定要确认其后是否跟着一个整数 n。

不难判断，第 32 行的填空（3）处需要填写的是对某个函数的调用（与第 47 行的判断文法相同）。但是这个函数是谁呢？如果我们注意文法的相同之处，结合函数 factor()的结构，不难判断这里是调用处理 δ 的函数，即函数 atom()。将 δ 的分子量计算出来赋给 num。注意，这里的判断还需要在以后进行确认，目前我们暂且这样认定。

从第 34 行开始，熟悉 C 语言的程序员都能够看出这是在判断 c 是否为一个数字字符。显然，如果是一个数字字符，就应该把它转换成数字。从第 41 行可以看出，转换以后的数字存储在变量 n 中。这样，第 36 行的语句就是完成字符到数字的转换的。而且从此处使用一个循环结构可以看出这里可以处理若干位的数字。从数字字符向数字的转换应该是很容易的，在这里，就应该描述为 ″n*10+c-'0'″。

回过头来仔细阅读这个函数，在实现程序功能方面没有潜在的逻辑错误，我们能够确认这样填写是符合要求的。

现在看函数 atom()。该函数是处理文法符号 δ 的。文法符号 δ 的文法规则是 δ→ζ|ζ θ|(λ)。不妨借助解读上两个函数的文法来解读该函数。从第 15 行到第 21 行处理了前两种情况，即 ζ 是一个大写字母或一个大写字母和一个小写字母。到第 22 行，从判断 c 是否为'('，可以看出这是处理第 3 种情况了。如何处理呢？第 23 行是一个函数调用（判断文法同上），由于在'('之后是一个 λ，显然，此后的函数调用应该是处理 λ 的函数，即函数 factor()。

这里的解题过程非常简单，即将程序与文法规则紧密结合，通过文法符号的处理来解题。首先要将程序中的语句与文法规则对应起来，然后就能够理解程序中的处理流程了。

[答案]

(1) factor()

(2) atom()

(3) n*10+c-'0'

(4) mAtom()或 factor()

(5) num+=d

试题 2 （2000 年试题 6）

阅读下列程序说明和 C 代码，将就填入（）处的字句在答卷的对应栏内。

[程序说明]

某城市有 n 个车站，并有 m 条公交线路连接这些车站，设这些公交车都是单向的，这 n 个车站被顺序编号为 0 到 n－1。本程序，输入该城市的公交线路数、车站个数和各个公交线路上的各站编号，求得从 0 站出发乘公交车到车站 n－1 的最少换车次数。

程序利用输入信息构建一张有向图 G（用邻接矩阵 g 表示），有向图的顶点是车站，若有某条公交线路经 i 站能到达 j 站，就在顶点 i 到顶点 j 之间设置一条为权 1 的有向边＜i,j＞。如果是这样，从站点 x 到站点 y 的的最少上车次数便对应图 G 中从点 x 到点 y 的路径长。而程序要求的换车次数就是上车次序减 1。

［程序］

```
#include <studio.h>
#define M 20
#define N 50
int a[N+1]; /* 用于存放一条线路上的各站编号 */
int g[N][N]; /* 存储对应的邻接矩阵 */
int dist [N]; /* 存储站 0 到各站的路径 */
int m,n;
void buildG()
{
int i,j,k,sc,dd;
prinft("输入公交线路数\n");
scanf("%d%d",&m,&n);
for(i=0;i<n;j++) /* 邻接矩阵清 0 */
for(j=0;j<n;j++)
g[i][j]=0;
for(i=0;i<m;i++)
{
prinft ("沿第条公交线路并进方向的各站编号
(0<=编号<=%d,-1 结束);\n",i+1,n+1);
sc=0; /* 当前线路站计算器 */
while(1)
{
scanf("%d",&dd);
if(dd= =-1)
break;
if(dd>=0 && dd<n)
(1)
}
a[sc]=-1;
for(k=1;a[k]>=0;k++) /* 处理第 i+1 条公交线路 */
for(j=0 ; j<k ; j++)
g (2) =1;
}
}
int minLen()
{
int j,k ;
```

```
for (j=0; j<n; j++)
dist [j]=g[0][j];
dist[0]=1;
do{
for (k=-1,j=0;j<n;j++) /* 找下一个最少上车次数的站 */
if (dist[j]>0 && (k= =-1 | | dist[j]<dist[k])) k=j;
if(k<0 || k==n-1) break;
dist[k] = -dist[k]; /*设置 k 站已求得上车次数的标志*/
for(j=1;j<n;j++)/*调整经过站能达到的其余各站的上车次数*/
if( (3) && (dist[j] == 0 || dist[k]+1 < dist[j]))
dist[j] = (4) ;
}while(1);
j=dist[n-1];
return (5) ;
}
void main()
{
int t;
buildG();
if((t=minLen()) < 0) printf("无解!"\n);
else printf("从 0 号站到%d 站需要换车%d 次\n",n-1,t);
}
```

[解析]

阅读程序说明，其中的第二段说明了两件事情：（1）有向图 G 中的数据是如何说明两站之间有无公交线路的，"若有某条公交线路经 i 站能达到 j 站，就在顶点 i 到顶点 j 之间设置一条权为 1 的有向边<i,j>"；（2）如何确定最少换车次数，"从站点 x 至 y 站点的最少上车次数便对应图 G 中从点 x 至点 y 的最短路径长度。而程序要求的换车次数就是上车次数减 1"。

综合程序结构，可以看到程序由 4 各部分组成：（1）文件包含和全局数据定义；（2）用于建立有向图 G 的函数 buildG()；（3）用于求出从站点 0 到站点 n-1 最少换车次数的函数 minLen()；（4）主函数 main()。

结合注释，我们可以从第 1 部分中明确 3 个数组的含义，由此确定他们在程序中的应用。数组 a[N+1]用于存放一条公交线路上的个站编号；数组 g[N][N]存储对应的邻接矩阵；dist[N]用于存储站点 0 到各站点的最短路径。下面我们来分析函数 buildG()。

在函数中用到的变量有 m、n、i、j、k、sc、dd 等。由第 10 行和第 11 行可以看出，m 表示公交线路数，n 表示公交站数；由第 17 行可以看出，sc 为当前线路站点计数器；其余 i、j、k 的不过用作循环变量而已。至于 dd 的用途，一时还难以确定。

结合注释部分阅读程序，从第 10 行到第 13 行都是很好理解的。从 14 行起的 for 循环，结束于第 27 行，构成本函数的主体，而且此后即是函数的结束，由此可以判定该循环完成输入各条线路并建立有向图的任务。

建立有向图，应该包括两个步骤，一是输入一条路径上的各个顶点，二是将输入路径的信息反映到邻接矩阵中去。将以上两个步骤套在一个循环中，反复执行即可完成有向图的建立。

研读第 14 行至第 27 行的 for 循环，可知在处理完 m 条公交线路后结束循环。分析循环体，这部分的结构是清晰的，除第 15、16、17 和 23 行外，由两个内层循环构成，即第 18 行至第 22 行的 while 循环和第 24 行到 26 行的 for 循环。

先研究 while 循环。由第 15、16 和 17 行与输入语句可知该循环完成一条公交线路上各个站点的输入。从第 19 行可知，dd 表示的是一个公交站点的编号。跳过第 20 行，我们看到这里有的"dd>=0 && dd<n"条件，这进一步印证了上面我们关于 dd 的判断。那么，既然 dd 是一个站点编号，在输入站点编号之后，我们首先要将其记入公交线路 a[ ]中，这里只有填空（1），别无语句，我们只好在这里填写完成该任务。同时还要注意到，作为当前线路站计数器，在 while 循环中没有对其进行修改的语句，所以在记入一个公交站点时，应相应的修改 sc。鉴 sc 的初值是 0，我们必须在这里填写"a[sc++]=dd"，如果使用"++sc"将会使 a[0]无值，也就是说公交线路的第一个站点将被遗漏。

借助注释，不难确定从第 24 行到 26 行的 for 循环完成将该条公交线路站点信息反映到邻接矩阵中的任务。考虑到公交线路是单向运行的，所以这里的操作也就是使公交线路上任意站点到其后任意站点之间存在一条权值为 1 的有向边。这样就需要使用一个二重循环，通过外层循环依次处理公交线路上的各个站点，通过内层循环将在站点前各站点与该站点之间设置一条为权 1 的有向边。

很明显，第 26 行就是完成上述内层循环任务的核心语句，这里应该怎样表述呢？从试题中我们可以看到填空（2）需要填写的应该是数组 g[ ] [ ]的下标。结合上面的分析，我们在这里填写"[a[j]] [a[k]]"是很自然的。这里一定要注意公交线路是单向运行的。

可能有的读者要对第 24 行的"k=1"表示困惑，因为这样便失去了处理公交线路的第一个站点 a[0]的机会了。这里仍旧要强调公交线路的单向运行。因为在单向运行的公交线路上，通过该线路上的任何站点都不能到达该线路的第一个站点，所以在循环中就直接从"k=1"开始，跳过线路的第一个站点。

下面我们来研读函数 minLen()。该函数中包含了 3 个填空，值得我们投入更多的精力来解答。
关于求有向图顶点到顶点之间最短路径，Dijkstra 有一个标准的算法，我们先在这里描述这个算法，然后结合这个算法来理解程序。

设指定的顶点为 v，把图中顶点集合分成两组，已求出最短路径的顶点集合为第一组，设为 S，其余的尚未最短路径的顶点集合为第二组。按最短路径长度递增

次序逐个把第二组中的顶点移入 S 中，直至从顶点 v 出发可以到达的顶点都在 S 中。在这个过程中，需要始终保持从 v 到 S 中各个顶点的最短路径都不大于从 v 到第二组中任何顶点的最短路径长度。另外，为了便于程序处理，要为每个顶点保存一个距离。S 中的顶点距离就是从 v 到此顶点的最短路径长度，第二组中的顶点的距离就是从 v 到此顶点的只包括以 S 中的顶点为中间顶点的那部分还不完整的最短路径长度。

根据以上描述，结合对程序的理解，就不难理解第 38 行到第 40 行的循环所处理的内容了，即调整经过站能到达的其余各站的上车次数。具体分两种情况：1、当存在从站 k 到站 j 的直通公交线路时，如果已经暂时确定了从站 0 到站 j 的最少上车次数(dist[j]>0)，在这种情况下，如果从站 0 经站 k 换车到站 j 的上车次数小于已经确定的从站 0 到站 j 的最少上车次数,则应该将经过站 k 到站 j 的上车次数(-dist[k]+1)确定为目前从站 0 到站 j 的最少上车次数。2。当存在从站 k 到站 j 的直通公交线路时，如果从站 0 到站 j 没有直通公交线路(dist[j]==0)，应该取经过站 k 到站 j 的上车次数(-dist[k]+1)为目前从 0 站到站 j 的最少上车次数。

将以上的分析翻译为 C 语言语句，那么填写（3）和填空（4）的解答就非常明显了，它们分别为"(dist[j]>=0 && g[k][j]= =1"和"-dist[k]+1"。细心的读者要问，在第 39 行中已经给出"dist[j]=0"的判断了，在填空（3）中还有必要再反映这一点吗？这是有必要的，如果我们在填空时不反映这一点，那么当 dist[j]==0 时将不进行调整处理，显然是不符合要求的。既然有了要填空的"dist[j]==0"，第 39 行中已经有的那个"dist[j]==0"还有存在的必要吗？答案是肯定的，这里就不再详细解释了，读者可以自己理解。

至于填空（5），不过是函数最后的回送返回值的语句而已。结合第 48 行和第 49 行，发现返回值有两种情况：1、无解（t<0）；2、有解（t>0）。显然在填空（5）中要处理好这两种情况。
要处理好这两种情况，必须从 k 的值入手。我们看第 36 行，可知在循环结束时 k 的值有两种情况，一是 k<0，二是 k=n-1。显然，k<0 是程序无解的标志。那么我们在这里就应该在 k<0 让函数返回一个负数，否则就返回从站 0 到站 n-1 的 0 最少换车次数（dist[n-1]-1）。因为在第 42 行中已经将 dist[n-1]赋给 j，所以我们可以构造填空（5）的答案为"k<0?-1:j-1"。

解答完本题，我们应该吸取的经验就是要切实掌握各种算法，同时要掌握各种算法在实际中的应用，通过程序语句与算法来解答题目。在过去几年中，最后一道 C 语言程序填空试题几乎都是关于回溯法的程序，其实也是关系到算法的掌握和应用。切实掌握了算法的实现，就可以非常容易地将程序（或函数）划分为几个部分，各部分都实现算法中的某一步骤。这样，结合试题的具体情况就能够快速写出正确的答案。

[答案]

（1）a[sc++] = dd

（2）[a[j]][a[k]]

（3）dist[j] >=0 && g[k][j] == 1

（4）-dist[k]+1

（5）k<0?-1:j-1

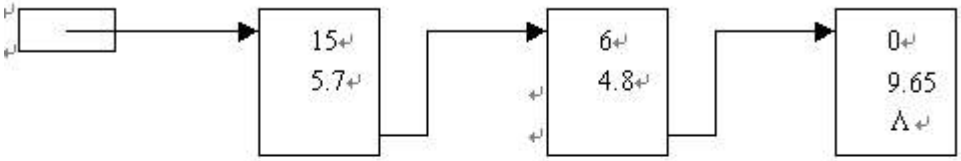
试题 3（1999 年试题 5）

阅读下列程序说明和 C 代码，将应填入（n） 处的字句写在答卷的对应栏内。

[程序说明]

本程序实现两个多项式像乘。多项式用链表表示，链表上的各表元按多项式的幂指数降序链接。例如：

$$f(x)=5.7x^{15}+4.8x^{6}+9.65$$



设两个多项式 $f(x)$ 和 $g(x)$ 分别为：

$$f(x)=f_n x^n+\Lambda+f_1 x+f_0 \qquad g(x)=g_m x^m+\Lambda+g_1 x+g_0$$

其积多项式为 $s(x)=f(x)g(x)=s_k x^k+\Lambda+s_1 x+s_0$

其中 $k=n+m, si=\sum_{u+v=i} f_u * g_v \quad (0\leqslant i\leqslant k)$

[程序]

```
#include <stdio.h>
#include <malloc.h>
typedef struct elem{int index;double coef;struct elem *next
}POLYNODE;
void write(POLYNODE *g)
{ POLYNODE *p=g;
```

```
while(p){ printf("%8.4f",p->coef);
if(p->index) printf("*^%d",p->index);
if(p->next && p->next->coef>0) printf("+");
p = p->next;
}
printf("\n\n");
}
main()
{ POLYNODE *f,*g,*s,*inpoly(),*polymul();
f=inpoly();g=inpoly();s=polymul(f,g);write();
}
POLYNODE *reverse(POLYNODE *g)
{ POLYNODE *u=NULL,*v=g,*w;
while(v){ w=v->next;v->next=u;u=v;v=w;}
return u;
}
POLYNODE *polymul(POLYNODE *f, POLYNODE *g)
{ POLYNODE *fp,*gp,*tail,*p=NULL,*q;
int i,maxindex; double temp;
maxindex = f->index+g->index; g=reverse(g);
for(i=maxindex;i>=0;i--){
fp=f;gp=s;
while(fp!=NULL &&fp->index>i) fp = fp->next;
while(gp!=NULL && gp->index<i-fp->index) gp = gp->next;
temp = 0.0;
while(fp && gp)
if(fp->index + gp->index ==i){
temp +=fp->coef*gp->coef; fp=fp->next; gp=gp->next;
else if( (1) ) fp=fp->next;
else gp=gp->next;
if(temp!=0.0){
q=( POLYNODE*)malloc(sizeof(POLYNODE));
q->index=i; q-coef=temp; q-next=NULL;
if( (2) ) p=q;
else (3) ;
tail =q;
}
}
g=reverse(g);
return p;
}
POLYNODE *inpoly()
{ POLYNODE *u,*v,*h=NULL,*p;
int index; double coef;
```

```
printf("Input index(<0 for finish)"); scanf("%d",&index);
while(index >=0){
printf("input coef"); scanf("%lf",&coef);
p=( POLYNODE*)malloc(sizeof(POLYNODE));
p->index=index; p->coef = coef;
v=h;
while(v!=NULL &&index<v->index) { nu=v;v=v->next;}
if(v==NULL || index > v->index)
{
p->next=v; if(v==h) (4) ;else (5) ;}
else v->coef +=coef;
printf("Input index(< 0 for finish)"); scanf("%d",&index);
}
return h;
}
```

[解析]

这道试题看起来很麻烦，其实不然，程序中科学地使用 3 函数，使程序的结构非常清晰。函数 inpoly()完成输入一个多项式各项的指数和系数，并建立链表的任务。函数 polymul()完成两项式的相乘的任务。现在我们就来分别研究这两个函数。

先看函数 inpoly()，它在输入多项式每一项指数和系数的过程中建立链表。语句"returnu;"完成回送返回值的操作，所以要特别函数中对 h 的操作。

一直到语句"p->coef;"是完成一个结点的输入，此后的"v=h"，因为"v=NULL"，所以这以后又有许多关于 v 是否为 NULL 的判断，足见 v 和 h 的变化对程序功能实现有极大关系。

接下来是一个循环体，如果仔细理解"index<v->index"这个循环条件的含义，就不难确定这是在"v= =NULL"的时候在中寻找当前输入结点的合适位置，以建成一个指数从大到小的链表形式的多项式。那么"v= =NULL"又代表什么？从程序中看，"v= =NULL"有两种情况：1、第一次输入，链表还没有建立；2、在寻找该结点的合适位置时遍历鲐链表的最后一个结点，即需要插入的结点应该在链表的末尾。

有了以上的理解，不难确定，以下 if-else 体中"index>v->index"的含义就是要在当前结点之前插入该结点。而"v->coef +=coef"则是完成在指数相等时的系数相加。仔细研究 if-else 体中的 if 部分，不难发现，该部分完成一个结点的插入，或者是在第一个结点输入后的处理。

先考虑第一次输入的情况，"p->next = v;"使 p->next 置空。填空（4）显然是处理第一次输入的情况（也包括当前输入结点指数大于链表中任何结点指数，该结点需要插入到链首的情况，读者可自己思考）。我们不管填空（5）的内容，

当再次执行到"v=h;"和其后的循环体时，不难发现在（4）中要将当前链表的首指针赋给 h，否则这个 while 循环就不能完成任务。那么(4)的内容显然是"h=p"。权衡上述各种情况，可以确定即使在当前输入结点指数大于链表中任何结点指数的情况下，程序仍能够完成任务。

基于上述的分析，不难确定填空（5）应完成在链表中或链表尾插入结点的功能。已经有了"p->next =v;"，这里只要填写"u->next=p"就可以完成任务了。

现在回到函数 polymul。首先，程序取两个多项式中最大指数的和为乘积的最大指数，然后调用函数 reverse()将链表 g 反转，使 g 的各个结点指数从链首到链尾由小到大进行排列。关于函数 reverse()的理解，这里就不多说了。

进入 for 循环之后的两个 while 循环让人摸不着头脑，但如果将 i=maxindex 和 gp->index<i-fp->index 以及 fp->index+gp->index>=i 联系起来则不难看出，这是在两个多项式 gp 和 fp 中寻找两个结点，这两个结点相乘后的指数（index）先于或大于当前正在处理的指数（当然不会有大于 maxindex 的，但在程序处理过程中两结点指数和大于当前正在处理指数的机会会有很多）。
显然，在找到一对指数和先于当前正在处理指数的时候，这是通过指数的相加和系数的相乘来生成一个由 q 来表示的新结点。这个结点应该插入链表中去，这就是填空（2）和（3）所在的 if-else 体应该完成的任务了。

这里的插入操作由一个 if-else 体和语句"tail=q"来完成。显然，插入时 p 有空和非空两种可能的状态，如果在（2）填写"p!=NULL"，那么"p=q"的结果是非常危险的，因为这将丢掉以前的运算结果，使我们最后只能悼念到运算结果的最后一项。因此这里只能填写"p==NULL"。

由此就可以确定（3）处理 p!=NULL 时的插入操作。由"tail=1"来看，tail 永远指向链尾的结点，那么这里应该先将 q 接到 tail 后，于是我们确定在（3）中填写"tial->next=q"。

现在只剩下填空(1)没有解决了。只有在两种情况下才能进入由"while(fp&&gp)"引导的循环：1、"gp->index+fp->index>i"；2、"gp->index+fp->index==i"。这里应该处理这两种情况，但为什么还要用一个 if-else 体呢？如果考虑 gp->index+fp->index==i 在进的处理我们可以看到 ，fp 和 gp 各自抽后移动了一个结点。fp 是按结点指数从大到小连接的，而 gp 是从小到大连接的，在处理了一次"gp->index+fp->index==i"的情况之后，很容易出现 gp->index+fp->index>i 或 gp->index+fp->index<i 的情况。当 gp->index+fp->index>i 时，为使其趋向 gp->index+fp->index == i，应该使 fp 向下移动一个结点反之使 gp 向下移动一个结点（为什么这样，请读者自己思考）。有了以上的理解，填空（1）的解答就显得很容易了，即 "gp->index+fp->index>i"。

[答案]

（1）fp->index+gp->index>i

（2）p == NULL

（3）tail->next=q;

（4）h=p

（5）u->next =p

试题 4（1999 年试题 6）

阅读下列程序说明和 C 代码，将应填入 （n） 处的字句写在答卷的对应栏内。

[程序说明]

本程序从 n 种不同重量、不同价值的物品中选取一部分物品。要求在不超过限定重量 limw 的前提下，使被选取的那些物品的总价值较大。这里约定 limw 不超过 n 种物品的重量总和,也没有一种物品的重量超过 limw，并且各物品的价值都大于 0。

程序中，n 种物品被顺序编号为 0、1、2、.....、n-1。

【程序】

```
#include <stdio.h>
#define N 100
double limw;
int opts[N]; /* 存储临时最佳的选择方案，当 opts[i]为 1，物品 i 在解中 */
struct elem { double weight;
double value;
} a[N]; /* 物品的重量和价值信息 */
int k, n ;
struct { int flg; /* 物品的考虑状态：0：不选，1：将被考虑，2：曾被选中
*/
double tw; /* 已达到的总重量 */
double tv; /* 期望的总价值 */
} twv[N]; /* 当前候选解中各物品的考虑状态，以及候选解的状态 */
main()
{ double maxv, find();
printf("Enter number of matter. "); scanf("%d", &n);
printf("Enter limit of weight. "); scanf("%lf", &limw);
printf("Enter weight and values of matters. ");
for (k = 0; k < n; k++) scanf("%lf%lf", &a[k].weight, &a[k].value);
maxv = find(a,n);
```

```
for(k = 0; k < n; k++) if(opts[k]) printf("%4d", k);
printf("\nTotal value = %lf\n", maxv);
}
next(int i , double tw, double tv) /* 将考虑 i 号物品 */
{ twv[i].flg = 1; twv[i].tw = tw; twv[i].tv = tv; }
look(int i, int *f, double *tw, double *tv) / * 取 i 号物品在解中的状
态信息 * /
{ *f = twv[i].flg; *tw = twv[i].tw; *tv = twv[i].tv; }
double find (struct elem *a, int n )
{ int i, k, f;
double maxv, tw, tv, totv = 0.0;
maxv = 0;
for(k=0; k < n; k++) ____(1)____;
next(0, 0.0, totv);
i = 0;
while(i >= 0) {
look(i, &f, &tw, &tv);
switch (f) {
case 1: twv[i].flg++; /* 先考虑被选中 */
if(____(2)____ <= limw) /* 选中可行吗？ */
if(i < n-1) { /* 后面还有物品吗？ */
next(____(3)____); /* 置 i+1 物品的状态 */
i++;
}
else if (tv > maxv) { /* 是一个更好的候选解吗？ */
maxv = tv;
for(k = 0; k < n; k++)
opts[k] = twv[k].flg != 0;
}
break;
case 0: i--; break; /* 回退 */
default: /* f == 2 */
twv[i].flg = 0;
if (____(4)____) /* 不选 i 号物品可行吗？ */
if(i < n-1) { /* 后面还有物品吗？ */
next(____(5)____);
i++;
}
else {
maxv = tv - a[i].value;
for(k = 0; k < n; k++)
opts[k] = twv[k].flg != 0;
i--;
}
```

```
break;
}
}
return maxv;
}
```

[解析]

仔细阅读程序说明和程序，不能发现试题中需要解答的问题全部集中在函数find( )中。在理解了函数 next( )和函数 look( )之后，我们开始阅读函数find( )。

首先，由于后面有关于 next( )调用时参数表的填空，应该研究函数 next( )的3 个参数的含义。阅读的 next( )函数体，能够确认：

参数 i 代表物品编号；

参数 tw 代表当前已经达到的总重量；

参数 tv 代表期望的总价值。

填空（1）之前有"totv=0.0"，之后 totv 作为函数 next( )的一个参数使用。通过研究函数 next( )，发现 totv 的值传给了 tw[i].tv，其含义应该是期望的总国。，显然 totv 不能是 0.0，那么（1）应该是对 totv 的值操作。究竟应该怎样赋值呢？根据说明，我们能够理解期望的价值应该人总价值开始试探，那么作为 for 循环的循环体，（1）应该主用来计算总价值的。所以我们在这里填写"totv+=a[k].value"。

现在看下面的程序。填空（2）所在的语句是通过 if 语句来判断选中某个物品是否可行，这里是考虑选中 a[i]是否可行。不可行的条件只有一个，即 a[i]的weight 与当前已达到的总之和超过了重量限制 limw。通过对函数 look( )的解读，可以知道程序中 tw 即表示当前已经达到的总重量，那么结合后面的"v<=limw"，这里应该填写"tw+a[i].weight"。继续往下阅读程序，包含填空（3）的语句及注释和此前的语句表明这里对函数 next( )的调用是在第 i 个物品已经被选中后，考虑第 i+1 个物品。

由于是第 i+1 个物品，因此这里的第 1 个参数就是 i＋1。第 2 个参数，因为第 i个物品已经被选中，所以已经达到的总重量就应该是"tw+a[i].weight"；第 3个参数是期望的总价值，在经过函数 next( )和 look( )的操作之后 ，tv 的值即为期望达到的总价值。在没有超过 limw 的情况下还可以继续选择物品，这表明该期望值是合理的，所以应该维持原来的数值，即 tv。所以（3）的正确答案为"i+1, tw+a[i].weight,tv"。

由"if(tv>maxv)"引导的 if 条件句是处理这样一种情况，即选中当前物品会使总重量超过 limw，从而在不选择物品 i 的情况下得到一个解。由此不难确定，maxv

记录了程序处理过程中达到的最大价值的数值。有必要说明一下，因为得到的解的总价值是各不相同的，所以最大的值就被记录下来，这里被选中物品的编号也就被记录在数组 ops[ ]中作为最佳方案。

现在看 CASE 结构的 default 分支。在这里，填空（4）所在的 if 语句判断不选物品 i 是否可行。若选中物品 i 也不能够使总价值达到期望总价值，就可以不选物品，那么（4）的解答可以这样构造，即"tv-a[i].value>maxv"。同时，在这种情况下应该修改期望的总价值，上述情况表明当前期望的价值是不合理的。

填空（5）要求解答的是在不选物品 i 的情况下考虑下一个(i+1)物品。参照填空（3）的解答，第 1 个参数就是 i＋1。那么第 2 个参数呢，这里决定不选物品 i 了，已经达到的总重量中就不能将物品 i 的重量包括进来，仍旧是 tw。第 3 个参数比较麻烦，物品 i 未选中，而且即使加上 i 的价值，当前最大价值仍旧不能够达到期望价值，就只好考虑修改期望价值，合理的修改方式是 tv-a[i].value。

另外，由于函数 next( )和 look( )使变量 tw 和 twv[i].tw、tv 和 twv[i].tv 具有等值关系，那么在解题时凡是使用的，都可以替换为 twv[i].twltwv[i].tv。

[答案]

（1） totv+=a[k].value

（2） tw+a[i].weight

（3） i+1,tw+a[i].weight,tv

（4） tv-a[i].value>maxv

（5） i+1,tw,tv-a[i].value

(注：解答中的 tw 可答为 twv[i].tw，tv 可答为 twv[i].tv).

试题 5 （1998 年试题 5）

阅读以下程序说明和 C 程序，将应填入(n)处的字句，写在答卷的对应栏内。

【程序说明】

这里给出的程序逐一从指走课程成绩文件中读入学生的学号和成绩，对同一学生汇总他的总成绩，并按以下格式输出名次（按总成绩由高到低的顺序）、总成绩、同一名次的学生人数、同一名次学生的学号（按学号由小到大的顺序）。

程序约定学生学习课程不超过 30 种，课程成绩文件的第一个数字就是课程号，统计过程中，同一课程号的成绩文件不能重复输入。

程序采用链表结构存储学生有关信息，链表中的每个表元对应一位学生。程序在数据输入过程中，形成一个按学生学号从小到大顺序链接的有序链表。当数据输入结束后，程序按总成绩从高到低，学号从小到大的顺序对链表排序。程序最后按指定格式输出链表中的信息。程序的输出格式如下例所示：

名次	总成绩	人数	学号
1	470	2	1225
3	450	3	1524
6	430	1	14
7	401	3	1318

【程序】

```
#INCLUDE〈STDIO.H〉
#DEFINEM30
#DEFINENLEN10
TYPEDEFSTRUCTNODE{INTCUR_S/*最近输入成绩的科目*/
                    CHARNO[NLEN]; INTSCORE;
                    STUCTIONNODE*NEXT;
                      } NODE;
NODE*BUBBLESORT(NODE*HEAD)
{NODE*Q, *TAIL, *P=(MODE*)MALLOC(SIZEOF(NODE);
  P→NEXT=HEAD; HEAD=P; TAIL=NULL;
  WHILE(TAIL!=___(1)___)
  {P=HEAD;Q=P→NEXT;
    WHILE(Q→ENXT! =TAIL)
    {IF(P→NEXT→SCORE→〈Q→NEXT→SCORE||
        P→NEXT→SCOR==Q→NEXT→SCORE&&
        STRCMP(P→NEXT→NO, Q→NEXT→NO)〉0)
      {P→NEXT=_(2)_, /*两相邻表元链接关系前后颠倒*/
        _(3)_=Q→NEXT→NEXT; P→NEXT→NEXT=Q;
      }
      P=P→NEXT; /*调整 P 和 Q*/Q=_(4)_;
    }
    TAIL=Q;
  }
  P=HEAD→NEXT; FREE(HEAD); RETURNP;
}
INTS[M], SP;
MAIN()
{FILE*FP;
  NODE*H, *U, *V*P;
```

```
INTSS, , MARK, ORDER, C;
CHARFNAME[80], NO[NLEN], ANS;
FOR(H=NULL, SP=0; ; )
{PRINT(''输入科目成绩文件名(输入 AAAA 表示强行结束)。\N");
  WHILE(1)
  {SCANF("%S",FNAME);
    IF(STRCMP(FNAME, "AAAA")==0)BREAK;
    IF((FP=FOPEN(FNAME, "R"))==NULL)
      PRINTF("不能打开文件%S, 请重新输入科目文件名。\N, FNAME);
    ELSEBREAK;
  }
  IF(STRCMP(FNAME, "AAAA")==0)BREAK;
  FSCANF(FP, "%D", &SS); /&MIDDOT;输入科目号&MIDDOT;/S[SP]=SS;
  FOR(I=0; S[I]! =SS; I++);
  IF(I<SP)
  {PRINTF("该科目的成绩已输入, 请输入别的科目成绩文件。\N");
    CONTINUE;
  }
  SP++;
  WHILE(FSCANF(FP,"%S%D", NO, &MARK)==2)
  {/*在链表中寻找最近输入的学号*/
    FOR(V=H;V!=NULL,&&STRCMP(V→NO,NO)<0;U=V,V=V→NEXT);
    IF(V! =NULL&&STRCMP(V→NO, NO)==0)
    {IF(V→CUR_S! ==SS)
      {V→SCORE+=MARK; V→CUR_S=SS;
      }/*同一科目成绩的重复输入, 后输入成绩被忽略*/
    }ELSE{P=(NODE*)MALLOC(SIZEOF(NODE); /*一位新的学生*/
          STRCPY(P→NO, NO); P→SCORE=MARK; P→CUR_S=SS;
          P→NEXT=V;
          IF(V==H)H=P; ELSEU→NEXT=P;
  }FCLOSE(FP);
  PRINTF("还有科目成绩文件要输入吗? (Y/N)"); SCANF("%C",&ANS);
  IF(ANNS=='N'||ANS=='N')BREAK;
}
H=BUBBLESORT(H);
PRINTF("名次总成绩人数学号\N"); /*以下按格式要求输出*/
V=H; ORDER=1;
WHILE(V! =NULL)
{FOR(C=1,U=V→NEXT;U!NULL&&U→SCORE==V→SCORE;C++,U=U→NEXT);
  PRINTF("%4D%7D%8D", ORDER, V→SCORE, C);
  FOR(ORDER+=C, I=1; _(5)_; V=V→NEXT, I++)
  {IF(I>1&&I%5==1)PRINTF("\N%23C", '');
    PRINTF("%S", V→NO);
  }PRINTF("\N");
```

```
    }
}
```

[解析]

本题考查的重点是数据结构和排序算法。虽然数据结构中的链表和排序算法中的冒泡排序属于比较容易掌握的内容，但把两部分结合起来运用应有一定的难度。这道题目给我们启示就是在复习中不要把各个知识点孤立起来，要达到融会贯通的程度，这样在考试中就不会发生费很多力气接受程序算法的现象。我们看到，本题的题干部分给出了很多注释，使我们能够比较轻松地理解程序。

不要为题干的篇幅所迷惑，如果借助注释阅读程序，就会发现一直到语句"h=bubblesort(h)"都出现需要填写的空白。本题需要填写的空白集中在链表排序和统计结果的打印上。

遵循程序的模块划分，首先进行填空（5）的填写。结合对以上程序的理解，易知需要打印的对象是一个有序链表，先按成绩排序，在成绩相同的情况下按学号排序。打印的格式要求已经给出，关键是如何实现按格式要求输出。语句"while(v!=NULL)"引导程序进入一个大循环，由循环条件"v!=NULL"和该循环的结束即程序的结束可以断定所有的输出均在循环体内完成。

仔细推敲循环体，发现循环体是由 3 个部分构成的。第 1 部分是一个 for 循环，适用于计算取得相同成绩的学生的个数，把计算结果存放在整型变量中，对这个循环的理解，关键是还要忽视了语句最后有一个"；"，如果忽视了这个"；"，对程序的理解就会出现根本性的错误。第 2 部分是一个打印语句，包括条打印名次(order)、总成绩(v-score)和取得该成绩的学生人数(c)。第 3 部分也是一个 for 循环，打印取得该成绩的学生的学号，我们重点研究这一部分。

从对第 1 部分的理解我们可以知道，c 为取得该成绩的人数，v 为取得该成绩的第一个学生结点，u 为取得该成绩的最后一个学生结点。理解了上述 3 个变量的含义，填空（5）循环条件的解答也就非常容易了，即尚未打印完毕取得该成绩的所有学生。表达方式是多种多样的，可以通过对取得该成绩的学生人数(c)递减来判断。这种判断的构造也是多种多样的，可以通过循环计数变量 i（在程序中主要用来控制打印学号的换行）和变量 c 的关系来构造，也可以通过判断结点 v 和结点 u 是否重合来构造，但最简单的特征条件应该是通过变量 i 和变量 c 的比较来构造。

现在回过头来看链表的排序。如果我们对"bubble"这个英语单词不陌生的，就应该马上意识到这是一种冒泡排序。函数中的第一句注释"两相邻表元链接关系前后颠倒"也显示出很强的冒泡色彩。函数的重心是一个二重循环，在这个二重循环中集中了我们需要解答的 4 个填空语句，所以对该循环的理解就是对本题的解答。

在几个变量初始化语句之后，进入循环体刚进入循环就遇到第 1 个填空，需要我们填写循环条件，显然在我们对该循环的循环体没有充分理解之前是难于完成这

个任务的，我们暂且跳过该语句，因为该循环是洽谈室要继续下去的，否则本题所提供的程序是无法工作的。循环体的前两条语句使我们得到这样的信息，结点 p 在前，结点 q 在后，每次执行这两条语句问题使结点 p 和结点 q 处于链表的前两个结点上。

接下来是内循环。内循环的一开始是一个 if 语句，该语句的判断条件很长，但仔细阅读，发现这是排序的条件－－－成绩高靠前，相同成绩时学号靠前。借助注释，我们能够与日俱增当相邻表元链接关系前后颠倒的条件成立时，需要颠倒的两个表元是 p->next 和 q->next 而不是 q 和 p。所以 p->next 就应越过 q，指向 q->next，q->next 应该指向 q->next->next。在 p->next->next 指向 q 之后，原来的 q 已经变为 p->next->next，原来的 q->next 已经变为 p->next。在一次比较（交换）之后，应该调整 q 和 p，在"p=p->next"语句之后，是对 q 的调整，显然，根据我们以上的理解，q 应该是 p 的后续结点，所以填写"q=p->next"是非常轻松的事情。

内循环结束之后，我们看到调整语句是"tail=q"，而每次比较（或交换）时，head 和 tail 是不参加的，这就意味着每次循环使比较（或交换）的对象减少一个。当若干次循环之后，和成为相邻表元，中间再也没有未经排序的表元之后，表示排序工作已经结束，循环也就应该结束，所以填空（1）的循环条件也就非常容易构造，即"tail!=head->next"。

[答案]

（1）head->next

（2）q->next

（3）q->next

（4）p->next 答 q->next 给 1 分

（5）c-- !=0 或 c-- 或 u!=v 或 i<=c 或 c-- >0

试题 6（1998 年试题 7）

阅读以下程序说明和 C 程序，将应填入(n)处的字句，写在答卷的对应栏内。

【程序说明】

本程序的函数 SUM(INT，IINTTOTAL，INTSIGMA，INTREAR，INTD[]，INTN)用来从已知数组 D 的前 N 个元素中找出所有部分元素序列之和等于 TOTAL 的元素序列，约定数组 D 的元素都是正整数，且都小于等于 TOTAL。

函数 SUM 使用递归方法找出全部解答。参数 I 表示递归函数当前考虑元素 D[I]，参数 SIGMA 是调用前已选取的部分序列的元素和，参数 REAR 是后面还未考虑的那部分元素的元素和。

函数对元素 D[I]有两种可能的选择方案：

1．考虑元素 D[I]被包含在新的部分元素序列中的可能性。如果在当前部分元素序列之后接上 D[I]，新序列的元素和不超过 TOTAL，则函数将 D[I]包含在当前部分元素序列中。如果新的部分元素序列的元素和等于 TOTAL 时，新的部分元素序列就是一个解答，函数将其输出；否则，若继续考虑后面的元素还有可能找到解答时，函数就递归去考虑后面的元素，寻找解答。最后，函烽就恢复原来部分元素序列中不包含 D[I]的状态。

2．考虑元素 D[I]不被包含在新的部分元素序列中的可能性。如果继续向 D[I]之后考虑还是有希望能得到和为 TOTAL 的部分元素序列时，函数将新序列不包含 D[I]也作为一种可能的选择，并递归去考虑后面的元素，寻找解答。

【程序】

```
＃INCLUDE 〈STDIO.H〉
＃DEFINE N100
INT A[N];
INT FLG[N];
SUM(INTI, INTTOTAL, INTSIGMA, INTREAR, INTD[], INTT)
{INTJ;
  /*考虑元素 D[I]被包含在新的部分元素序列中的可能性*/
  IF(SIGMA＋D[I]TOTAL/*如果 D[I]与当前序列的和不超过 TOTAL*/
   {FLG[I]＝1; /*D[I]被考虑在被部分元素序列中*/
    IF(_(1)_＝＝TOTAL)
    {/*输出解*/
      FOR(J＝0; FLG[J]＝＝0; J＋＋);
      PRINTF("％4D＝％D", TOTAL, D[J]);
      FOR(J＋＋; ＜＝I; J＋＋)
          IF(FLG[J])
              PRINTF("＋％D",D[J]);
      PRINTF("\");
    }
    ELSE/*并且继续考虑后面的元素有可能找到解答时*/
    IF(I＜N－1&&REAR－D[I]＋SIGMA＞＝TOTAL)
        SUM(I＋1, TOTAL, _(2)_, REAR－D[I], D, N);
    _(3)_;
  /*考虑元素 D[I]不被包含在新的部分元素序列中的可能性。*/
  IF(I＜N－1&&REAR－D[I]＋TIGMA＞＝TOTAL)
          SUM(I＋1, TOTAL, _(4), REAR－D[I], D, N);
}
```

```
MAIN()
{INTI, J, N, TOTAL, S, D;
  PRINTF("输入 TOTAL! /N"); SCANF("%D",&TOTAL);
  PRINTF("输入 N! /N"); SCANF("%D",&N);
  FOR(S=I=0; I<N; )
  {PRINTF("输入第%D 个元素>0 且<=%D)\N", I+1, TOTAL);
    SCANF("%D", &D);
    IF(D<1||D>TOTAL)
      {PRINTF("出错, 请重新输入! \N");
          CONTNUE;
      }
      S+=A[I++]=D;
  }
  SUM(0, TOTAL, 0,_(5)_, A, N);
  PRINTF("\N\N");
}
```

[解析]

如果不仔细阅读程序说明,可能会话中有话话中有话误以为本题考查的重点是探测法,但在切实理解了程序说明之后,我们能够认识到本题的考查重点是函数的递归调用。我们仍旧强调要结合程序说明来解答题目,因为题干所提供的程序是对程序说明的踏实实现,或者说题干所提供的程序就是程序说明的另一种表达方式。

在仔细阅读程序说明之后,从 main( )开始读起。在读完程序的 for 循环之后,可以得到如下信息;整型数组 a 中有个 n 元素,是选取符合要求的元素序列的范围; s 为数组 a 中所有元素的和; total 为我们选取的特定元素序列的和,凡是和为 total 的元素序列都将被视为符合条件的元素序列加以输出。for 循环之后是对函数的调用,其中第 4 个实际参数是要解答的问题。参照程序说明,发现该参数为后面还未考虑的那部分元素的元素和。在这里,函数是第一次调用,函数当前考虑的元素是 a[0],调用前已选服部分序列的和为 0,则该填空处应该填写的内容就是数组 a 所有元素的和。结合对 for 循环的理解,正确答案应该是。此处易犯的错误是填写"s-a[0]"。一定要记住,该参数是尚未包含进序列后面的那些元素的和。这一点也可以从填空(2)和填空(4)所在的两次函数调用得到佐证,因为我们看到只有在确定了 d[i]是否包含进序列中之后才在 rear 中减去 d[i]。很明显,此处尚未将 a[0]包含进序列,故只能填写"s"。

下面我们对本题的主体部分——函数 sum()进行解析,解答包含在函数体内的 4 个填空。函数体的主要内容是两个 if 语句,仔细阅读选择数组元素的方案,发现第一个 if 语句对应第 1 种选择方案,第 2 个 if 语句对应第 2 种选择方案。现在就可以把整个函数体分成两个互不相干的两部分,分别加以考查。

第 1 个 if 语句的内容饱含个部分,即 1、将 d[i]包含进部分元素序列中; 2、if-else-if 语句,该语句实现了选择方案 1; 3、需要解答的填空,该语句在

if-else-if 语句条件不成立时执行。很明显，填空（1）处判断某个值是否等于 total，结合下面的注释和对程序的理解，能够很容易持理解在该判断成立时输出一个符合条件的元素序列，因此就不难确定这里判断考虑进 d[i]之后元素序列的和是否等于 total。未考虑进 d[i]之前的元素序列的和是 sigma，那么考虑进 d[i]之后的元素序列之和就应该是 sigma+d[i]了，我们应该毫不犹豫地将"sigma+d[i]"填写到答题纸上。

填空（2）是函数调用　的第 3 个实际参数，结合注释和程序说明，很容易就能推断出该函数调用实现了在 d[i]被包含进元素序列之后而元素和尚小于 total 的处理。根据心目对程序的理解，易知第 3 个实际参数应该是包含进 d[i]之后的元素序列和，未包含 d[i]之前的序列的和是，那么包含进 d[i]之后的元素序列之和就应该是 sigma+d[i]，因此此处应该填定与填空（1）相同的内容。

如果你觉得想像不出填空（3）应该填写什么答案，那么就再回过头去读程序说明中的选择方案。这里有一句话就是对填空（3）的表述－－"函数应恢复名不包含的状态"。把 d[i]包含进序列的操作是把数组 flg 的第 i 个元素赋值为 1，而且在上面打印符合要求的元素序列时跳过在 flg 中对应值为 0 的元素。易知某个元素不包含在序列中时，在 flg 中对应的元素值应该为 0，即恢复序列中不包含 d[i]的状态，就应该是 flg[i]的重新置 0 了，所以此处应该填写"flg[i]=0"。函数 sum()主体的第 2 个 if 语句要实现的功能是，当元素 d[i]不被包含在序列中时，继续求解。既然元素序列中没有包含 d[i]，则元素序列的和就没有发生变化，仍旧是上次调用时的值，即 sigma。至此我们已经解答了全部问题。

回过头来看看本题的解答，我们应该坚定这样的钥匙思路，把程序划分成相对独立的几部分，给各部分在程序说明中找到对应的描述。但难度也就在这里。如果程序划分或与程序说明对应时发生失误，可能会导致我们的解答全车覆没，只要大家在平时的练习中注意这方面的锻炼，发生错误的可能性还是比较小的。

[答案]

（1）sigma+d[i]

（2）sigma+d[i]

（3）flg[i]=0

（4）sigma

（5）s

试题 7（1997 年试题 6）

阅读以下程序说明和 C 程序，将应填入_(n)_处的字句，写在答卷的对应栏内。

【程序说明】

某系统由 N 个部件组成，这些部件被物理地分成若干个分离的部件组。同一组内的两件部件 I 和 J，它们或直接相连，或间接相连（部件 I 和部件 J 间接相连是指在这两件部件之间有一个部件相连序列，其中部件 I 和 J 分别与这相连序列中的某个部件直接相连）。系统的 N 个部件被统一编号为 0，1，…，N-1。本程序输入所有直接相连的部件号对，分别求出系统各分离部件组中的部件号并输出。

程序根据输入的直接相连的两件部件号，建立 N 个链表，其中第 I 个链表的首指针为 S[I]，其结点是与部件 I 直接相连的所有部件号。

程序依次处理各链表。在处理 S[I]链表中，用 TOP 工作链表重新构造 S[I]链表，使 S[I]链表对应系统中的一个部件组，其中结点按部件号从小到大连接。

【程序】

```
#INCLUDE
#DEFINE N 100
TYPEEFSTRUCTNODE{
INTDATA;
STRUCTNODE*LINK;
}NODE;
NODE *S[N];
INT I,J,N,T;
NODE *Q,*P,*X,*Y,*TOP;
MAIN()
{PRINTF("ENTERNUMBEROFPARTS.");
SCANF("%D",&N);
FOR(I=0;IDATA=J;P->LINK=S[I];S[I]=P;
P=(NODE*)MALLOC(SIZEOF(NODE));
P->DATA=I;P->LINK=S[J];S[J];=P;
}
FOR(I=0;IDATA]!=NULL)
{/将 J 链表也移入工作链表*/
FOR(P=S[J];P->LINK! =NULL;P=P->LINK);
P->LINK=TOP;TOP=S[J];__(3)__;
}
/*在重新生成的第 I 链表中寻找当前结点的插入点*/
FOR(Y=S[I];__(4)__;X=Y,Y=Y->LINK);
IF(Y!=NULL&&Y->DATA==Q->DATA)
FREE(Q);/*因重新生成的第 I 链表已有当前结点，当前结点删除*/
ELSE{/*当前结点插入重新生成的 I 链表*/
__(5)__;
IF(Y==S[I])S[I]=Q;
ELSEX->LINK=Q;
```

```
}
}
FOR(I=0;IDATA);
Q=P->LINK;FREE(P);P=Q;
}
PRINTF("\N");
}
}
```

[解析]
本题考查的知识要点是链表操作。链表的基本操作是简单的，仔细阅读试题说明，结合程序中的说明文字，不难发现本题的难度主要在最后一段中，"在处理链表 s[i] 表中，用 top 工作链表重新构造 s[i] 表，使 s[i] 链表对应系统中的一个部件组，其中结点按部件号从小到大连结。"

阅读试题所提供的程序，我们能够从输入部分的构造链表算法中推知，这里建立的是循环链表，即链表的尾结点的指针指向链表头。结合程序说明，5 个需要回答的问题集中在顺序处理各个链表部分，所以这里的关键是要结合程序理解程序中使用的算法。

仔细推敲程序，可以断定处理各链表的过程应该为：对由 s[i]所指向链表，将该表移到由指针 top 所指的工作链表。对 top 链表的各结点进行如下处理：从 top 链表上取出一个结点，根据该结点所指出的相连部件，将由 s[j]所指向的链表也移入 top 链中，并将取出的结点按部件递增顺序重新构造由 s[i] 所指向的链表，如此重复，直到 top 链表为 NULL。这样由 s[i] 所指向链表的重新构造就结束了。所有链表处理完后，重新构造好的各非空链表，与系统中的各部件组一一对应。

仔细阅读填空（1）所在的语句，还要忽视填空（1）前是"，"，而不是"；"，可以推知，这里应该是将 s[i] 置空，因为该循环的任务是重新构造链表。结合心目输入部分的链表构造及以后对由 top 所指向的链表的使用，不难推知由 top 所指向的链表也是循环链表，结合上下文，填空（2）的答案应该是" top=top->link "，实现链表尾结点的指针指向链表头的操作。

考查填空（3）及其前面的两条语句，它们所承担的任务与" top=s[i] "和填空（1）所承担的任务应该是相同的，由彼及此，得出（3）的应该是很容易的。

填空（4）所在的语句组完成在重新链表中寻找当前结点的插入点的任务。根据试题说明，链表是按部件号递增顺序构成的，结合基本的链表遍历知识，（4）的答案应该是"y!=NULL && y->data<q->data "。根据填空所在语句及程序说明，结合链表插入操作的基本流程和程序对链表构造的要求，答案应该是"q->link=y"。

[答案]

（1）S[i] = NULL

（2）Top = top->link

（3）S[j] = NULL

（4）y !=NULL && y->data <q->data 或 y && y->data <q->data

（5）q->link = y

试题 8（1997 年试题 8）

阅读以下程序说明和 C 程序，将应填入_(N)_处的字句，写在答卷的对应栏内。

【程序说明】

一个相连的区域被不规则地分割成 N 个不同的小区域;每个小区域与若干其它小区域相邻接。现用 CN 种不同颜色为该区域着色，要求每个小区域着同一种颜色，相邻小区域着不同颜色。

设小区域被顺序编号为 0，1，…，N-1。每个小区域与其它小区域的邻接关系用两维数组 BORDERING 表示，元素 BORDERING[I][J]表示 I 号小区域与 J 号小区域之间的邻接关系：

BORDERING[I][J]=0 J 小区域与 I 小区域不邻接
BORDERING[I][J]=1 J 小区域与 I 小区域相邻接

程序中，把计算结果存入于两维数组 COLORED 中，颜色编号为 0,1,…,CN-1,元素 COLORED[COLER][J]的含义是

COLORED[COLOR][J]=0 J 小区域不用颜色 COLOR 着色
COLORED[COLOR][J]=1 J 小区域用颜色 COLOR 着色

函数 COLORCOUNTRY(BORDERING,COLORED,N,CN)根据所给的小区域邻接关系数组 BORDERING、小区域个数 N、颜色数 CN，将找到的着色方案记录在数组 COLORED 中。函数采用试探法找解。首先从第一个小区域着第一种颜色开始顺序为各小区域找着色方案。对某个小区域，当为它找到一种未被它的相邻小区域着色的颜色时，就用该颜色对该小区域着色，并准备处理下一个小区域。当不能为某个小区域找到一个未被它的相邻小区域着色的颜色时，就回溯。如最终为全部小区域找到着色方案，函数返回 1；否则，函数返回 0。

程序假定小区域个数不超过 20，颜色数为 4。

【程序】

```
#INCLUDE
#DEFINE N 20
#DEFINE CN 4
INT
COLORCOUNTRY(INTBORDERING[][N],INTCOLORED[][N],INTN,INTCN){INTCOLOR,U
SED,I,C;
FOR(COLOR=0;COLORN;I++)COLORED[COLOR][I]=0;
C=0;/*从第 1 个小区域开始*/
COLOR=0;/*从着第 1 种颜色开始试控*/
WHILE(CC;I++)
IF(__(2)__)USED=1;
IF(!USED)BREAK;/*当前颜色未被相邻小区域着色*/
COLOR++
}
IF(!USED)
{/*找到一种可用颜色，用此色着色，并准备处理下一个小区域*/
__(3)__=1;COLOR=0;
}ELSE{/*未找到一种可用颜色，回溯*/
C--;IF(C<0)RETURN0;/*发现没有解的情况
*/FOR(COLOR=0;__(4)__;COLOR++);__(5)__=0}}RETURN1;}PRINT(INTCOLORED[]
[N],INTN,INTCN)/*输出结果
*/{CHAR*COLORT[]={"RED","BLUE","GREEN","YELLOW"};INTCOLOR,I;FOR(COLOR
=0;COLORN;I++)
IF(COLORED[COLOR][I])PRINTF("\T%D",I);
PRINTF("\N");
}
}
INTCOLORED[CN][N],BORDERING[N][N];
MAIN()
{INTC,I,J,N;
PRINTF("ENTERNUMBEROFAREAS.");SCANF("%D",&N);
PRINTF("ENTERBORDERING:\N");
FOR(I=0;IN;J++)BORDERING[I][J]=0;
FOR(I=0;IOTONEXT).\N",I};
SCANF("&D",&J);
WHILE(J>=0)
{IF(I!=J)BORDERING[I][J]=BORDERING[J][I]=1;
SCANF("%D",&J);
}
}
IF(COLORCOUNTRY(BORDERING,COLORED,N,CN))
PRINT(COLORED,N,CN);
```

```
ELSEPRINTF("NOSOLUTION.\N");
}
```

[解析]

该题考查的仍然是回溯法的问题。试题说明非常详细，对算法的注释也非常详尽。关键是我们是否能够将试题说明和程序结合起来读，将试题说明中的每一部分与程序中的语句段建立联系。程序中有许多注释，这使试题说明与程序的联系非常明显。

需要解答的问题全部集中在函 colorcountry( ) 数中，我们就集中精力解读这个函数。首先结合程序注释明确各个变量的含义，这是对以后程序理解的基础。然后从宏观上把握函数　结构，不难发现，函数体的主体部分是一个 while 循环。循环由一个 while 循环和 if-else 体构成。结合试题说明与程序注释，可以推知内层 while 循环对每一个小区进行着色，if-else 体将处理找到合适颜色和找到合适颜色的情况，把握了程序的结构，我们甚至不用对整修程序有透彻的理解，只要结合变量含义、试题说明与程序注释就可以解答问题。

首先，我们由填空（1）后面的注释可以推知这里应该是 color 与 cn 的比较，以控制循环的结束。既然是顺序对每种颜色作试探，则在每次执行到这里时，应该递加。因为循环体中已经实现了 color 的递加，所以（1）的解答就可以简单地构造为"color<cn"。填空（2）所在循环完成的任务是检查当前颜色是否已被某相邻小区域着色，结合循环中两个变化的变量 i、c 以及对试题说明的理解，当前颜色已被某相邻小区域着色，应该表示为 bordering[c][i]不为 0，colored[color][i]不为 0，所以填空(2)的答案就可以表示为"bordering[c][i] && colored[color][i]"。

填空（3）、（4）、（5）集中在处理找到或没有找到的体中。找到了如何处理？首先要在 colored[]中登记，即 colored[colored][c]将置 1。如果单从这一个处理来看，这样的答案也算是正确的结合注释中"准备处理下一个小区域"的语句，通观全局，准备工作自然不能在处理没有找到合适颜色时进行，那么我们在这里实现对 c 的递加，即将答案改为 colored[colored][c++]，注意 c++与++c 的区别，体会在这里这样填写的理由。

显然，填空（4）和（5）完成没有找到合适颜色时的处理。结合心上着色的算法推算回溯的算法，不难推出回溯的基本算法，即回溯过程中不但要回到前一步，还要能将当前已设置的信息恢复到它原告的状态，然后，以便可以根据前一步的状态继续试探其他方案。我们知道，数组 colored 中已经记录了将当前颜色要重新返回到上一个区域的颜色值。程序中（4）的解答为"colored[colored][c]"。回溯的过程除了恢复状态外，还要取消原有的选择，继续进行选择，得到下一个可能的值。因此填空（5）的内容应该为"colored[colored++][c]"。通过该语句，就未完成了回溯过程，可以继续篁，向前继续试探。

我们可以看到，本题的 5 个问题中只有（4）和（5）才涉及到回溯法的构造，其余的 3 个问题都是可以通过对试题说明和程序结构、功能的把握来解答的，这也

从另一个方面说明了在钥解题时仔细阅读试题说明和从结构上把握程序的重要性。

[答案]

（1）color < cn 或 color < 4

（2）bordering[c][i] && colored[color][i] 或 bordering[c][i] == 1&&colored[color][i]==1

或 bordering[c][i] * colored[color][i] == 1 其中 bordering[c][i]可写成 bordering [i][c]

（3）colored[color][c++]

（4）colored[color][c]=0 或! colored[color][c] 或 colored[color] != 1

（5）colored[color++][c]

试题 9（1996 年试题 5）

阅读以下程序说明和 C 程序，将应填入程序中_(n)_处的字句，写在答卷的对应栏内。

【程序说明】

本程序是一个简单的计算器模拟程序。对任意给定的正确四则运算表达式，程序计算其结果值并输出。表达式中运算分量为无正负号整数，运算符为+、_、*、/，圆括号按常规配对，表达式以字符"="结束。

函数 getach()为获取表达式的一个合法自负，并将字符存入变量 curch；函数指针树组 func[]是为了统一加减乘除计算而设置的。

【程序】

```
#include <stdio.h>
int add(int x,int y){return x+y;}
int sub(int x,int y){return x-y;}
int mul(int x,int y){return x*y;}
int div(int x,int y){return x/y;}
int (*func[])()={add,sub,mul,div};
int num,curch;
char chtbl[]="+-*/()=";
char corch[]="+-*/()=0123456789";
int getach()
```

```
{ int i;
while(1)
{ curch=getchar();
if(curch==EOF) return -1;
for(i=0;corch[i]&curch!=corch[i];i++);
if(i<strlen(corch)) break;
}
return curch;
}
int geid()
{ int I;
if(curch>='0'&&curch<='9')
{ for(num=0;curch>='0'&&curch<='9';getach())
num=_①_;
return -1;
}
else{for(i=0;chtbl[i];i++)
if(chtbl[i]==curch) break;
if(i<=5) getach();
return i;
}
}

int cal()
{ int x1,x2,x3,op1,op2,i;
i=getid();
if(i==4) x1=cal(); else x1=num;
op1=getid();
if(op1>=5) return x1;
i=getid();
if(I==4) x2=cal(); else x2=num;
op2=getid();
while(_②_)
{ i=getid();
if(i==4) x3=cal(); else x3=num;
if((op1/2==0)&&(op2/2==1))
x2=(*func[op2](x2,x3));
else{ x1=_③_;
x2=x3;
_④_;
}
op2=getid();
}
return _⑤_(x1,x2);
```

```
}
void main()
{ int value;
printf("Please input an expression:\n");
getch();
while(curch!='=')
{ value=cal();
printf("The result is : %d\n",value);
printf("Please input an expression:\n");
getach();
}
}
```

[解析]

本程序是一个简单的计算器模拟程序，对值得任给的正确的四则运算表达式，程序计算其结果值并输出。程序由一个主程序和数个子程序组成。主程序的功能比较简单。输入一个表达式，计算其结果并输出，然后输入下一个表达式。4 个计算子程序 add、sub、mul、div 和输入字符子程序 getach 都很简单，这里不再浪费笔墨。

getid 子程序的功能是判断刚才输入的字符是操作数不是运算符。程序使用了一个 if 语句，若读入的字符是'0'到'9'之间的，则是操作数，那么继续将该操作数全部读进来（因为操作数也是以字符方式读进来的，因此一个操作数需要分为多个字符读进来，然后再装配成一个操作数），并将其值保存至全局变量 num 中。读入字符的顺序是先高位后低位，因此读入的数字字符要按照由高位到低位的顺序依次转化为一个数，所以填空（1）的答案为"num*10+curch-'0'"。若读入的字符不是数字,那么要判断一个是哪个运算符,不同的运算符返回不同的值。

cal 子程序是本题的关键所在。它比较复杂，使用了递归算法。它的功能是：首先判断已经输入的字符，若是'('则根据括号优先的原则，先递归调用自己，求出括号里的表达式的值，否则是操作数（本程序无查错功能，假定所输入的都是正确的四则运算表达式，这样一个表达式的开头不是括号就肯定是操作数），将其值从全局变量 num 赋给 x1。接着读入一个运算符，若运算符是')'或'='，则完成一级运算，返回结果 x1。读入第 2 个操作数，同上，若读入的字符是'('，则根据括号优先的原则，先递归调用自己，求出括号里的表达式的值，否则将其值赋给操作数 x2。然后读入第 2 个运算符。下一步的操作应当是先读入第 3 个操作数，再根据 op1 和 op2 的优先级决定进行运算 op1 还是 op2，并将计算结果作为操作数 x1 或 x2，继续后计算。循环的结束条件是遇到')'或'='，说明一对括号内的计算完成整个计算结束。因此，填空（2）的答案为"op2<5"。在循环内部，先读入第 2 个操作数，同上，若是'('则先求括号里的表达式值。如果第一个运算符 op1 为'+'或'-'，第 2 个运算符 op2 为'*'或'/'，那么先进行 op2 并将计算结果记入 x2 中；否则先进行 op1 并将结果放入 x1 中，x3 放入 x2 中，op2 放入 op1 中，以进行下一步运算。因此，填空（3）的答案为"(*func[op1])(x1,x2)"，（4）的答案为"op1=op2"。整个循环结束后，只剩下

x1、x2 和 op1，那么最后计算一下 x1、op1 和 x2，作为整个表达式的结果，因此填空（5）的答案为"(*func[op1])(x1,x2)"。

[答案]

（1）num*10+curch-'0' 其中'0'可答成 48 或 0x30 或 corch[7]

（2）(op2>=0)&&(op2<5) 或 op2<5 或 op2<=4 或 !(op2>=5)

（3）(*func[op1](x1, x2)

（4）op1=op2

（5）(*func[op1])

试题 10（1996 年试题 7）

阅读下列程序说明和 C 程序，将应填入程序中(n)处的字句，写在答卷纸的对应栏内。

【程序说明】

本程序先从文件读入各考生的准考证号（设为整型数）及成绩，并将其存放在一棵检索二叉树上，二叉树的结点的键值是成绩，二叉树每个节点带一链表，链表结点存放取得该成绩的考生的准考证号。然后，程序按中序遍历检索二叉树，从高到底分输出结果，使每行输出某成绩及其取得该成绩的各考生的准考证号。

【程序】

```
#include <stdio.h>
typedef struct idnode{
int id;
struct idnode *next;
}IdNode;
typedef struct marknode{
int mark;
IdNode *head;
struct marknode left,right;
}MarkNode;
char fname[]="sp07.dat"'
main()
{ int id,mark;
MarkNode *root=null;
```

```
FILE *fp=fopen(fname,"r");
if(!fp){
printf("file %s open error.\n",fname);
exit(0);
}
while(!feof(fp)){
fscanf(fp,"%d%d",&id,&mark);
btree(&root,id,mark);
}
fclose(fp);
print(root);
}
btree()
{ IdNode *ip;
MarkNode *mp=mpptr;;
if(_①_){
if(mark==mp->mark) addIdNode(_②_,id);
else if(mark>mp->mark) btree(&mp->left,id,mark);
else btree(&mp->right,id,mark);
}else
{ mp=(MarkNode *)malloc(sizeof(MarkNode));
mp->mark=mark;
mp->left=mp->right=NULL;
_③_
addIdNode(&mp->head,id);
_④_
}
}
addIdNode(IdNode **ipp,int id)
{ IdNode *ip=ipp;
if(_⑤_) addIdNode(_⑥_,id);
else{
ip=(IdNode *)malloc(sizeof(IdNode));
sp->id=id;
ip->next=NULL;
_⑦_
}
}
print(MarkNode *mp)
{ IdNode *ip,ip0;
if(mp){
print(mp->left);
printf("%6d: \t",mp->mark);
ip=mp->head;
```

```
while(ip){
printf("%6d: \t",ip->id);
ip0=ip;
ip=ip->next;
free(ip0);
}
printf("\n");
print(mp->right);
free(mp);
}
```

[解析]

本程序着重考查对检索二叉树、链表的基本操作及双重指针的用法等知识点的掌握。

二叉树或者为空，或者由一个根结点加上左子树和右子树（互不相交的两棵二叉树）组成。检索二叉树左子树的每一个结点的值都小于（或大于根结点的值，右子树的每一个结点的值都大于（或小于）根结点的值，其左子树和右子树也是检索二叉树。双重指针是指向指针的指针，它的值是一个指针的地址。

大致浏览一个程序，程序主要由一个主程序和 3 个函数 btree、addidNode、print 组成。主程序的功能比较简单。打开一个文件，循环读出文件中的每一条记录，并将该记录的内容存放在一棵检索二叉树上，所有的记录读完以后，再将该检索二叉树打印出来。

程序最重要的地方是在函数 btree 中，btree 的功能是将一个新的标点插入检索二叉树中，它有 3 个参数，mpptr 是检索二叉树的根结点，id 和 mark 是从数据文件中读入的准考证号和成绩。

检索二叉树的插入算法是若二叉树的根结点为空，则创建一个根结点，给其赋上相应初值，否则判断根结点的键值与所给的值的大小。结果可能有 3 种：所给值大小根，则递归检索根的左子树，即在左子树上插入新的结点；所给值等于根结点值，结点已存在；所给值小于结点值，则递归检索根的右子树，即在右子树上插入新的结点。

根据检索二叉树的插入算法，分析 btree 的算法。题中 if(1) { }else { }的结构，对应于检索二叉树的插入算法中判断根结点是否为空的算法。因而填空（1）应为"mp!=NULL"。若根结点不空，则：若所给 mark 与根结点的 mark 值相等，就需要在根结点的 id 链表上增加一个结点，调用函数 addidNode 实现这一功能，因此，填空（2）所在的语句为"addidNode(&mp->head,id)"；若所给 mark 大于根结点的 mark 值，则需要将该记录插入在根结点的左子树上，递归调用函数 btree 实现这一功能；若所给 mark 小于根结点的 mark 值，则需要将该记录插入在根结点的右子树，递归调用函数 btree 实现这一功能。若根结点为空，则创建一个根结点，给其赋上相应初值，赋给其 mark 值，初始化其左右子树为空，初

始化其 id 链表为空，所以填空（3）答案为"mp->head=NULL"。此时，插入工作 id 尚未完成，该考生尚未保存，再调用函数 addldNode 将该考生的 id 插入到该结点的 id 链表中。最后，应将临时指针 mp 赋回给根结点，所以填空（4）的答案为"*mpptr=mp"。

函数 addidNode 与 btree 有些类似，该函数也是一个递归函数，它与 btree 的不同在于它是在一个线性链表上操作，而 btree 是在一个检索二叉树上操作。addIdNode 的功能是在一个链表尾部插入一个结点。它的算法也是先判断结点是否为空，若不空，则需递归在其 next 结点上插入，所以函数 addidNode 中填空（5）的答案为"ip!=NULL"，（6）的答案为"addIdNode(&p->next,id)"，即将结点插入到当前链表的下一结点所带链表的后面；若为空，则需要新建一结点，初始化其 next 为空，赋给其 id 值。若 ip 为空，并将临时指针 ip 赋回给原来的指针，所以填空（7）的答案为"*ipp=ip"。

程序最后根据已经建好的检索二叉树打印排序好的结果，方法是中序遍历树上每个结点，结每个结点依次访问结点 id 链表中的每一个结点，打印出其 id 值。

[答案]

（1）mp 或 mp !=NULL 或 *mpptr 或 *mpptr !=NULL\

（2）&mp->head 或 &(mp->head)

（3）mp->head=NULL

（4）*mpptr=mp

（5）ip 或 ip !=NULL 或 *ipp 或 *ipp !=NULL

（6）&ip->next 或 &(ip->next)

（7）*ipp=ip

试题 11（1995 年试题 7）

阅读下列程序说明和 C 程序，将应填入程序中 (n) 处的字句，写在答卷纸的对应栏内。

【程序说明】

本程序用回溯算法来产生由 0 或 1 组成的 2^m 个二进位串，使该串满足以下要求。

视串为首尾相连的环，则由 m 位二进制数字组成的 2^m 个子序列，每个可能的子序列都互不相同。例如，如果 m=3，在串 11101000 首尾相连构成的环中，由 3 位二进制数字组成的每个可能的子序列都在环中恰好出现一次，它们依次是 111、110、101、010、100、000、001、011，如图所示。



图 8-1 1995 年试题 7

【程序】

```
#define N 1024
#define M 10
int b[N+M-1];
int equal(int k,int j, int m)
{ int i;
for(i=0;i<m;i++)
if(b[k+1] (1) ) return 0;
return 1;
}
int exchange(int k, int m, int v)
{ while(b[k+m-1] == v)
{ b[k+m-1] = !v; (2) ;}
(3) =v; return k;
}
init(int v)
{
int k;
for(k=0;k<N+M-1;k++) b[k] = v;
}
main()
```

```
{ int m,v,k,n,j;
printf("Enter m(1<m<10),v(v=0,v=1)\n");
scanf("%d%d",&m,&v);
n=0x01<<m; init(!v); k=0;
while( (4) < n)
for(j=0;j<k;j++)
if(equal(k,j,m))
{ k=exchange(k,m,v);
j= (5) ;
}
for(k=0;k<n;k++)
printf("%d\n",b[k]);
}
```

[解析]

本程序比较短，但处处都有填空，难度比较大，着重考查考生对试探回溯算法的掌握。

程序的功能是用回溯算法来产生由 0 或 1 组成的 2^m 个二进制字串，所以主程序首先输入 m 和 v，其中 m 是一个二进制字串的位数，v 为 0 或 1。由函数 init 可知，在主程序中先将要求的二进制字串初始化为!v，从而 v 只能为 0 或 1，v 为 0 表示将二进制字串初始化为 1，第 1 个 m 位二进制字串为 m 位 1；v 为 1 表示将要求的二进制字串初始化为 0，第 1 个 m 位二进制字串为 m 位 0。接着程序通过移位运算使 n 为 2m（其实就是一个乘幂运算，但用移位速度要快得多），n 是要填写的位数，即要填写的二进制字串的元素的个数；置 k 为 0，k 为最新填写的 m 位二进制字串在 b[ ]（b[ ]为所求的二进制字串）中的开始下标。在初始化时，自 b[0]开始，已填写了一个 m 位的二进制字串，其值为!v。k+1 也可看作已在 b[ ]中正确填写的二进制数的个数，然后是一个试探求解的循环。在已有 k+1 个二进制数满足要求，且 k+1<n 的情况下循环。一般地，试探循环应包括以下工作：

(1)增 1。

(2)在 b[k+m-1]处预填入试探的值（此处工作已由循环之前的函数完成，将其置为!v），表示一个新的 m 位二进制数字已填入。其高 m-1 位是前一个 m 位二进制数的低 m-1 位。

(3)检查新的 m 位二进制数是否与前面已经填写的 k 个 m 位二进制数相等。如都不相等，本次试探完成，继续下一次试探；如有相等的，就得改变最新填入得二进制数字的值，改变后，需重新检查。若仍不符合要求，则需要回溯，改变上一次试探的二进制数字的值，若仍不符合要求，则继续回溯，直至符合要求或无解。

在新一轮试探循环开始时，已经有 k+1 个正确的 m 位二进制字串，而本次循环的目的是测试第 k+2 个 m 位二进制字串是否符合要求，即与前 k+1 个 m 位二进制字

串是否不相等。如初始循环时 k=0，则已有一个正确的 m 位二进制数字（m 位全是!v），进入循环是测试从 b[1]开始的第 2 个 m 位二进制数字（m 位也全是!v）与第 1 个 m 位二进制数字是否相等。所以 while 后的条件表达式为++k<n，即天空（4）的答案为"++k"。

for 循环的功能是检查第 k+1（因为在 while 条件表达式中 k 值已加 1）个 m 位二进制数字串与前 k 个 m 位二进制字串是否相等。它首先调用函数 equal 判断第 k 个 m 位二进制字串是否与第 j 个 m 位二进制字串相等。函数 equal 比较两个 m 位二进制字串是否相等，采用的是逐位比较的方法，依次比较与 b[k+i]与 b[j+i]（0<=i<=m-1），若相等则继续比较下一位，直至最后一位仍相等，则这两个二进制字串相等，返回 1；否则，两个二进制字串肯定不相等，不用再进行下面的比较，直接返回 0 即可。所以填空（1）的答案为"!=b[j+i]"。若 equal 函数返回 1，即两个二进制字串相等，则说明这一步试探失败，此时调用 exchange 函数进行更改，包括可能的回溯，并返回更改（回溯）后的 k 值。更改（回溯）后，又要从第 1 个 m 位二进制字串开始检查第 k+1 个 m 位二进制字串是否与前面 k 个 m 位二进制字串的某个相等，即要让 for 循环重新开始（j=0）。因为 for 循环执行结束时，要先执行表达式 j++，为了使下次循环重新开始时执行，即执行 j++后，j 的值是 0，填空（5）处应置为-1。所以填空（5）的答案为"-1"。

函数 exchange 的功能是第 1 次试探失败后对 b[k+m-1]（b[k+m-1]为第 k+1 个 m 位二进制的最后一位）的值进行更改（包括回溯）。首先判断 b[k+m-1]的值，若为 v 则肯定 b[k+m-1]的值已经更改过了，必须进行回溯，此时应先恢复 b[k+m-1]的值为!v，然后 k 值减 1。继续进行判断，若前一位的值仍为 v，则继续回溯，直至其值不等于 v 为止才停止回溯，然后将其值改为 v，并返回更改后的 k 值。若为!v 则还没有更改过，将其值变为 v 后返回。所以填空（2）的答案是"k--"，或其他使 k 减 1 的 C 代码；填空（3）的答案是"b[k+m-1]"。
最后将求得得二进制字串打印。

[答案]

（1）!=b[j+1]

（2）k--或--k 或 k=k-1 或 k-=1

（3）b[k+m-1]

（4）++k 或 k=k+1 或 k+=1

（5）-1

试题 12（1994 年试题 7）

阅读以下程序说明和 C 程序，将应填入_(N)_处的字句，写在答卷的对应栏内。

[程序说明]

由二叉树的前序遍历和中序遍历两个遍历序列能唯一确定一棵二叉树。

前序遍历为：访问根结点、访问左子树、访问右子树。

中序遍历为：访问左子树、访问根结点、访问右子树。

如图 8-2 所示的二叉树，其前序和中序遍历序列分别为：

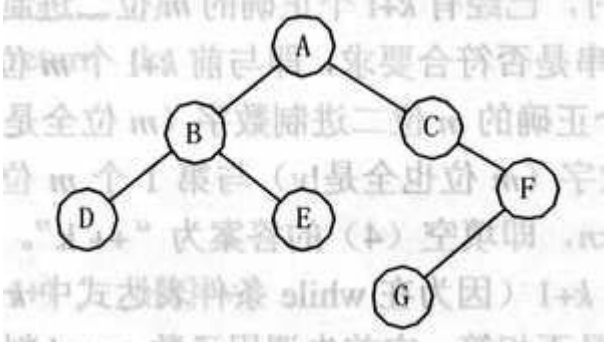


图 8-2 1994 年试题 7

pred[]：A、B、D、E、C、F、G。

inod[]：D、B、E、A、C、G、F。

本程序实现由已知某二叉树的前序遍历和中序遍历序列，生成一棵链接表示的二叉树。

构造二叉树的算法的要点是：由前序遍历序列，该序列的第一个元素是根结点元素（例中为 A）。该元素将中序遍历序列分成左、右两部分，那些位于该元素之前的元素是它的左子树上的元素（例中为 D、B、E），位于该元素之后的元素是它的右子树上的元素（例中为 C、G、F）。对于左、右子树，由它们的前序遍历序列的第一个元素可确定左、右子树的根结点，参照中序遍历序列又可进一步确定子树的左、右子树元素。如此递归地参照两个遍历序列，最终构造出二叉树。

两个遍历序列作为主函数 main()的参数。为简单起见，程序假定两个遍历序列是相容的。主函数调用函数 restore()建立二叉树。函数 restore()以树（子树）的前序遍历和中序遍历两序列及序列长为参数，采用递归方法建立树（子树）。

函数 postorder()实现二叉树的后序遍历序列输出，用来验证函数 restore()建立的二叉树。

[程序]

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct node{
char info;
struct node *llink,*rlink;
}TNODE;
char pred[MAX],inode[MAX];
TNODE *restore(char*,char *,int);
main(int argc, char **argv)
{
TNODE *root;
if(argc <3) exit(0);
strcopy(pred,argv[1]);
strcpy(inode,argv[2]);
root = restore(pred,inodstrlen(pred));
postorder(root);
prinft("\n\n");
}
TNODE *restore(char *ppos, char *ipos,int n)
{
TNODE *ptr;
char *rpos;
int k;
if(n<=0) return NULL;
ptr = (TNODE*)malloc(sizeof(TNODE));
ptr->info = (1) ;
for( (2) ;rpos<ipos+n;rpos++)
if(*rpos == *ppos) break;
k= (3) ;
ptr->llink = restore(ppos+1, (4) ,k);
ptr->rlink = restore( (5) +k, (6) +1,n-1-k);
return ptr;
}
oistirder(TNODE *ptr)
{
if(ptr == NULL) return;
postorder(ptr->llink);
postorder(ptr->rlink);
```

```
printf("%c",ptr->info);
}
```

[解析]

本程序的功能是已知某二叉树的前序遍历和中序遍历序列，生成一棵链接表示的二叉树。其依据是由二叉树的前序遍历和中序遍历序列能唯一确定一棵二叉树。前序遍历序列的第一个元素即为二叉树的根结点，同时该元素也把对应的中序遍历序列分成两个部分，左边是二叉树的左子树，右边是二叉树的右子树。前序遍历序列的第二个元素是二叉树左子树的根结点，它又可将对应的左子树的中序遍历序列分成两部分。如此递归下去，则能够确定每个结点再二叉树中的位置。

整个程序的结构很简单，由主程序和两个子程序组成。主程序根据输入的前序序列和中序序列，调用子程序 restore 来建立二叉树，然后再调用 postorder 子程序后序遍历二叉树并输出，以验证 restore 建立的二叉树，postorder 功能比较简单，下面着重讲一下 restore。

子程序 restore 以二叉树的前序序列*ppos、中序序列*ipos 及二叉树的结点个数 n 为参数，返回指向二叉树的指针。当 n 小于等于 0 时，反回空。否则先申请一个二叉树结点空间，并将对应的元素的值赋给该结点，与该结点对应的是前序序列的递一个元素，所以填空（1）的答案是"*ppos"或"ppos[0]"。查找该结点元素在中序序列的位置，查找是从前往后，以循环方式进行的，一旦找到就退出循环，所以填空（2）的答案为"rpos=ipos"。接下来应该是递归分析出该结点的左子树及右子树。这一步的关键是把正确的参数传给递归子程序。

观察程序可知，填空（4）应为该层二叉树的右子树的中序序列，（5）应为该层二叉树的右子树的前序序列，（6）应为该层二叉树的右子树的中序序列，（3）应为该层二叉树的左子树的结点个数。左子树中结点的个数，就是在中序序列中根结点之前的元素的个数。所以填空（3）的答案是"rpos-ipos"。在前序序列中元素的顺序是根结点、左子树的前序序列、右子树的前序序列，所以递归调用时左子树的前序序列的指针为 ppos+1，而右子树的指针为 ppos+1+左子树的结点个数。所以填空（5）的答案是"ppos+1"。在中序序列中元素的顺序是左子树的中序序列、根结点、右子树的中序序列，所以递归调用时左子树的中序序列为，而右子树的中序序列为 ipos+1+左子树的结点个数（或为 rpos+1，rpos 在循环结束时指根结点）。所以填空（4）的答案是"ipos"，填空（6）的答案是"rpos"或"ipos+k"。

[答案]

（1）*ppos 或 ppos[0]

（2）rpos=ipos

（3）rpos-ipos

（4）ipos 或 rpos-k

（5）ppos+1

（6）rpos 或 ipos+k

试题 13（1993 年试题 7）

阅读以下程序说明和 C 程序，将应填入_(N)_处的字句，写在答卷的对应栏内。

[程序说明]

对于正整数 n，输出其和等于 n 且满足以下限制条件的所有正整数的和式，即组成和式的数字自左至右构成一个非递增的序列。如 n=4，程序输出为：

```
4 = 4
4 = 3 +1
4 = 2+2
4=1+1+1+1
```

程序中给出了分别采用递归和非递归解法的两个函数 rd()和 nd()。

函数 rd()采用递归解法，它有两个参数 n 和 k。其意义分别是被分解和式的数 n，及当前第 k 深度分解。算法思想是对 n 的所有合理的和式分解，将分解出的数(称为和数）存于数组 a[ ]中。当其中一个分解已不再需要进一步分解时，即找到下一个解，将存于数组 a[ ]中的一个完整和式的和数输出。当还需要进一步分解时，以要进一步分解的数及分解深度为参数，递归调用分解和式函数。

函数 nd()以要分解的数为参数，另开设一个数组 r[ ]，用于存贮当前还未分解的答数。在求一个解的第 k 步时，a[k]为第 k 个和数，r[k]为相应的余数。当找到一个分解后（此步 r[k]等于 0），输出解，并作回溯处理，从当前 k 退回到第一个不为 1 的和数，将其减 1，并将其余数加 1，准备去找另一个解；否则，生成下一步的分解和数与余数。

[程序]

```
#define MAXN 100
int a[MAXN],r[MAXN]
rd(int n,int k)
{
int i,j;
for(j= (1) ;j>=1;j--)
{
```

```
a[k]=j;
if( (2) )
{
printf("%d=%d",a[0],a[1]);
for(i=2;i<=k;i++)
printf(+%d",a[i]);
printf("\n");
}
else (3) ;
}
}
nd(int n)
{
int i,k;
k=0;r[0]=n;
do
{
if( (4) )
{
printf("%d=%d",a[0],a[1]);
for(i=2;i<=k;i++)
printf(+%d",a[i]);
printf("\n");
}
while(k>0 && (5) ) k--;
if(k>0){
a[k]--;r[k]++;}
else{
a[k+1] = (6) ;
r[k+1]=r[k]-a[k+1];
k++;
}
}while(k>0);
}
int test_data[]={3,4,5};
main()
{
int i;
for(i=0;i<sizeof(test_data)/sizeof(int);i++)
{
a[0]=test_data[i];
rd(test_data[i],1);
printf("\n___________\n\n");
nd(test_data[i]);
```

```
printf("\n_____\n\n");
}
}
```

[解析]

本程序的功能是递归和非递归解两种解法，将给定的正整数 n 分解成为一个数字自左向右构成一个非递增序列的和式。主程序比较简单，只是用 3 组数据测试一下递归和非递归解两种解法的结果是否一致。本程序的关键在于递归和非递归解两种求解算法。

首先看一下采用递归解法的函数 rd( )，它有两个参数：被分解的正整数 n 和当前的分解深度 k。根据题意，要分解出来的和数序列构成一个非递增序列，因此需限制分解出来的数不能大于 a[k-1]，即对 n 分解时，最大数是 n 和 a[k-1]中较小的那个，所以填空(1)的答案是"n<a[k-1]?n:a[k-1]"。当从 n 分解出和数 j，将 j 存入 a[k]后，就可处理对余数 n-j 的分解。当 n-j 等于 0 时，即 j = = n 时，正整数 n 的分解宣告结束，a[1]至 a[k]就是 n 的一个完整的和式分解，将它们输出即可；否则，递归调用函数 rd( )对 n-j 作和式分解，此时被分解的数为 n-j，当前分解深度为 k+1。所以填空(2)的答案为"j = = n"或"a[k]= = n"；填空（3）的答案为"rd(n-j,k+1)"。

对于采用非递归解法的函数 nd( )，它只有一个参数，即要分解的正整数 n。根据试题的解法叙述，程序预设第 0 步分解的初值，k 为 0，r[0]为 n。求解过程是一个循环，直到回溯到第 0 步找出全部解。由程序说明可知，当前分解的余数 r[k]为 0 时，就找到了一个解，程序将存于数组 a[ ]中的和数按试题的格式要求输出，所以填空（4）的答案为"r[k] = = 0"。输出后，进行回溯调整处理，准备找下一个解，应当回溯至第一个不为 1 的和数为止，所以填空（5）的答案是"a[k] = = 1"。回溯后进行调整，将该步的和数减 1，余数增 1，程序已给出调整的全文。如当前分解的余数，则要确定下一步分解出的和数和余数。已知当前分解的和数为 a[k]，余数为 r[k]，下一步是对 r[k]进行分解。因为所求的和式是一个非递增的序列，所以分解出的最大和数应取 a[k]与 r[k]中的小者，所以填空（6）的答案是"a[k]<r[k]?a[k]: r[k]"。当回溯至 k = = 0 时，所有的分解都已求出，此时循环结束。

[答案]

（1）n<a[k-1]?n:a[k-1]

（2）j==n 或 a[k]==n

（3）rd(n-j,k+1)

（4）r[k]==0 或 r[k]<=0

（5）a[k]==1 或 a[k]<=1

（6）a[k]<r[k]?a[k]:r[k]

试题 14（1992 年试题 7）

阅读以下程序说明和 C 程序，将应填入_(N)_处的字句，写在答卷的对应栏内。

[程序说明]

（1)本程序利用辗转相除法求两个均不超过 100 次的多项式 A、B 的最大公因式。

例：

$A(x)=x^3-x^2+x-1=(x^2+1)(x-1)$

$B(x)=x^5-7x^4+7x^3-3x^2+6x+4=(x^2+1)(x^3-7x^2+6x+4)$

最大公因式为：x^2+1

（2）辗转相除法的算法如下：

用其中的一个多项式去除另一个多项式；然后，将所得余式变成除式，原除式变成被除式。如此反复相除，当余式为 0 时当前除式即为最大公因式。

[程序]

```
#include <stdio.h>
#include <math.n>
#define DECISE E.0005
#definde MAX_POWER 100
main()
{
int i,a,b;
float Ca[MAX_POWER+1],Cb[MAX_POWER+1];
void Remainder();
scanf("%d",&a);
for(i=0;i<=a;i++)
scanf("%d",&Ca[i]);
scanf("%d",&b);
for(i=0;i<=b;i++)
scanf("%d",&Cb[i]);
Remainder(Ca,Cb,a,b);
}
void Remainder(PointerA,PointerB,a,b)
```

```
float *PointerA,*PointerB;
int a,b;
{
float x,y,*Temp;
int i,j,Flag=1;
while(Flag)
{
i=0;
while(PointerB[i]==0)
{
i++;b--;
(1) ;
}
x=PointerB[i];
while(i<=b)
PointerB[i++] /=x;
for(i=0;i<=a-b;i++)
{
(2) ;
for(j=0;j<b;j++)
{
y = PointerA[i+1+j]-x*PointerB[j+1];
PointweA[ (3) ]=(y<DECISE && -y<DECISE)?0.0:y
}
}
Termp = PointerA;
PointerA = PointerB;
PointerB = (4) ;
a=b--;
for(Flag=0,i=0;i<b &&Flag==0;i++)
if(PointerB[i]==0.0) Flag=1;
}
printf("The Greatest Common Factor is:\n);
for(i=0;i<a;i++)
if(PointerA[i]=0.0)
printf("%5.3f*^%d%s",PointerA[i].a-i,(PointerA[i+1]<0.0)?"":"+");
printf("%5.3f\n",PointerA[a]);
}
```

[解析]

本题中函数 Remainder 用来实现求两个多项式的最大公因式。函数体内的"while(FLAG)"循环用来控制辗转相除过程。该循环中的"while(PointerB[i]==0"循环是求多项式 Cb[ ]中系数为非零项的最高幂次 b。根据后续程序的要求，

要使指针指向幂次为 b 的那一项，并使保持为 0。因此填空（1）为"PointerB+i--"。

双重 for 循环的功能是实现多项式相除，把余式保留在 Ca 中。内循环中每次使一个 PinterA[i]为，要把其系数赋值给 x，因此填空（2）应为"x=PointerA[i]"。考虑到两数相减，其差的绝对值可能很小，此时可视为 0。每次求得一个以后，都要做这样的判断，因此填空（3）应为"i+1+j"。

辗转相除后，余式比原 Ca[ ]降低了 a-b+1 次，所以余式系数的开始位置为"PointerA+a-b+1"，但若 a<b，前面的辗转相除并未行、执行，故填空（4）应为"(a>=b)?Temp+a-b+1:Temp"。

[答案]

（1）PointerB += i--;

（2）x = PointerA[i];

（3）i+1+j

（4）(a>=b)?Temp+a-b-1:Temp

试题 15 （1991 试题 7）

阅读以下程序说明和 C 程序，将应填入_(N)_处的字句，写在答卷的对应栏内。

[程序说明]

本程序用来对英文文献中出现的单词按下列计算公式统计其使用频度：

某单词的使用频度＝ 该单词的使用次数/文献中单词的总数

程序中用一个根结点为的分类二叉树存放文献中的每个英文单词及其在文献中的使用次数。对于二树中的每个结点中的单词，按字典序比较大小，小的存放在它的左子树中，大的存放在其右子树中。当统计完毕后，程序将按中序遍历该分类二叉树，得到下一个按英文单词字典分类的单词表（），并且按单词的使用次数为关键字降序输出统计结果。

程序调用了如下一些函数：

1、函数将某单词按二分查找法插入分类二叉树中，并统计该单词在文献中的使用次数。

2、函数按中序遍历分类二叉树，并形成相应的单词表中序遍历，即先访问其左子树，然后访问根结点，最后访问其右子树。

3、函数从存放英文文献的数据文件""中取出一个单词，放入字符数组中。如果遇到文件结束标志，则返回 0；否则，返回 1。为简单起见，假定每个英文单词长度不超过 20 个字母。

4、函数将单词表以单词的使用次数为关键字，按降序排序。

5、函数输出单词表中各单词和它的使用频度。

6、函数将字符串拷贝至字符串。

7、函数按字典顺序对两字符串和进行比较。倘若大于，则返回一个正整数；若等于，则返回 0；若小于，则返回一个负整数。

程序清单中省去了上述编号为 3、4、5、7 的函数的内容。

[程序]

```
#include <stdio.h>
#include <alloc.h>
#define EMPLY 0
#define MAXLEN 21
typedef struct tree
{ char word[MAXLEN];
int count;
struct tree *leftson;
struct tree *rightson;
}BINARY;
typedef struct
{
char word[MAXLEN];
int count;
}LIST;
BINARY *treeroot=EMPTY;
main()
{
FILE *fp,*fopen();
char keyword[MAXLEN];
int total_word=0;
fp=fopen("TEXT.H","r");
while(getword(keyword,fp))
{
binary_tree(keyword);
total_word++;
```

```
}
fclose(fp);
reault=(LIST*)malloc(total_word * sizeof(LIST));
travel_tree(treeroot,result);
sort_list(result,total_word);
display_list(result,total_word);
}
binary_tree(keyword)
char * keyword;
{
BINARY *ptr,*temp;
int strcmp_reault;
temp=EMPTY;
ptr=treeroot;
while(ptr=EMPTY)
{
strcmp_result=strcmp(keyword,ptr->word);
if(strcmp_reault==0)
{
ptr->count++;
return;
}
else
temp=ptr;
if(strcmp_reault>0)
ptr-> (1) ;
else
ptr = (2) ;
}
}
ptr=(BINARY *)malloc(sizeof(BINARY));
ptr->rightson=ptr->leftson=EMPTY;
ptr->count=1;
strcpy(keyword,ptr->word);
if(temp==EMPTY)
treeroot=ptr;
else
if(strcmp_reault>0)
temp-> (3) =ptr;
else
temp-> (4) =ptr;
}
travel_tree(root,result)
BINARY *root;
```

```
LIST result[];
{
static x=0;
if(root==EMPTY)
(5) ;
else
{
travel_tree( (6) ,result);
strcpy(root->word,result[x].word);
result[x++].count=root->count;
}
travel_tree( (7) ,result);
}
strcpy(s1,s2)/*copy string s1 to s2*/
char *s1,*s2;
{
while( (8) ='\0');
}
```

[解析]

本程序的功能是用来统计英文文献中单词的使用频度的。本题的程序比较长，乃至的函数也比较多，似乎很麻烦，其实不然。程序开始定义了一个二叉树的结点类型和一个线性表的结点类型。在主程序中，先从文件中逐个读入单词，并调用函数 binary_tree 将其插入到分类二叉树中，同时统计总单词量。然后，调用函数 bravel_tree 中序遍历生成的分类二叉树，并将结果放入线性表 result 中。接着调用函数 sort_list，按单词的使用次数对线性表 result 排序。最后，将排好序的二叉树打印出来。主程序比较简单，除了一个 while 循环外，其他语句都是顺序执行。下面我们看重分析函数 binary_tree 和 travel-tree。

首先看一下函数 binary_tree。binary_tree 的功能是将一个单词插入至分类二叉树中。分类二叉树，即排序二叉树中、，其插入算法是：

1、若二叉树为空，则申请一个结点，将关键字 keyword 赋给它；

2、若关键字大于根结点的关键字，则在右子树上插入；

3、若关键字小于根结点的关键字，则在左子树上插入。

对照程序，首先进行判断，若该结点不空，则比较 keyword 与根结点的关键字。若相等，即该单词已在二叉树中存在，则将该结点对应的 count 字段加 1，并返回；否则，保存指向该结点的指针（即 temp 指针指向下一次循环查找的结点）。若 keyword 大于该结点的关键字结点的指针（即 temp 指针指向下一次循环查找的结点）。若 keyword 大于该结点的关键字(strcmp_result>0)，则在其右子树上查找插入，融在其左子树上查找插入，下一步就以该结点的左子树（或右子树）

为新的开始点进行查找插入，一直到找到譔单词的结点或结点为空（该单词在二叉树中不存在）为止。所以填空（1）的答案是："ptr->rightson"，填空（3）的答案是"ptr->result>0"。若未在 while 循环中退出函数，则表明单词 keyword 不在二叉树中，需要将其插入。先申请一个结点空间，指针 ptr 指向刚申请到的空间地址，令 ptr 的左右子树为空，将其 count 字段.置为 1，将 keyword 赋给其 word 字段。若指针 temp 为空，则表明二叉树为空，将 ptr 赋给二叉树的根结点指针；若 temp 不为空，则由于 temp 指针指向下一次循环查找的结点，而在最后一次循环中结点 ptr 为空，所以在退出 while 循环后，temp 指针的左子树或右子树为空（由 strcmp_result 决定），即我们要插入的结点应该是 temp 的左儿子或右儿子。当 strcmp_result>0 时，要插入的结点应该是 temp 的右儿子，否则，要插入的结点应该是 temp 的左儿子。所以填空（3）的答案是"rightson"，填空（4）的答案是"lsftson"。

函数 travel_tree 按中序遍历该分类二叉树，并形成相应的单词表，将其存放于线性表 result 中。所谓中序遍历，即先访问其左子树，然后访问根结点，最后访问其右子树。二叉树的遍历一般采用递归算法。本函数就是采用递归来实现中序遍历分类二叉树。若当前结点为空则返回，否则先递归遍历其左子树，然后访问该结点，再遍历右子树。对照程序，显然，填空（5）的答案是"return"，填空（6）的答案是"root->leftson"，填空(7)的答案是"root->rightson"。静态变量 x 的作用是指示数组 result 的下标，用以确定在递归调用时结点存入数组 result 的位置。

函数 strcopy(s1,s2)的功能是将字符串 s1 拷贝至字符串 s2。正常的作法是用一个循环一个字符一个字符地拷贝，当遇到字符串结束符'\0'时结束。而此处只用一条语句就要实现这个功能，难度比较大。其实只要我们掌握了 C 语句里的指针和"＋＋"运算符，做这道题詘不是太难。填空（8）的答案为"(*s2++ = = *s1++)"。*s1++的值指 s1++的是之前的所指的字符，待取出后才改变 s1 的值，同理，先将字符存入 s2 所指的位置，再改变 s2 的值。当遇到字符串结束符'\0'时结束(此时'\0'也拷贝给了 s2），否则将把*s1++!='\0'的比较结果赋给*s2++。

[答案]

（1）ptr->rightson

（2）ptr->leftson

（3）rightson

（4）leftson

（5）return

（6）root->leftson

（7）root->rightson

（8）(*s2++=*s1++)

试题 16 （1990 年试题 7）

阅读以下程序说明和 C 程序，将应填入_(N)_处的字句，写在答卷的对应栏内。

[程序说明]

设对于一个 n*n 的上三角矩阵 a，为节约存储，只将它的上三角矩阵元素按行主序连续存放在数组 b 中。下面的函数 trans 在不引入工作数组的情况下，实现将 a 改为按列主序连续存放在数组 b 中。

设 n=5，

$$a=\begin{vmatrix}1&2&3&4&5\\0&6&7&8&9\\0&0&10&11&12\\0&0&0&13&14\\0&0&0&0&15\end{vmatrix}$$

b=(1，2，3，4，5，6，7，8，9，10，11，12，13，14，15)

经调用 trans 函数后，b 变为

b=(1，2，6,3，7,10,4，8,11,13,5,9，12,14,15)

函数 trans 对数组元素的存储位置用调整。调整过程中存在若干个循环传送链：b(i1)→b(i2)…b(ij) →b(i1) 1≤j<n

例如，考查调整后的数组元素 b(2)（值为 6），与该元素相减的位置调整将形成下面的循环传送链：b(2) →b(3) →b(6) →…b(12) →b(9) →b(5) →b(2)。

关键是确定循环传送链的下标 i1 ， i2 ， …ij ，以及在考查调整后的元素 b(k)(k=3,4,…)时能判定 b(k)是已传送过的某传送链上的元素。

函数 ctr(k,n)计算调整后的数组 b 的第 k 个元素 b(k)在原数组 b 中的位置，该位置作为函数 ctr(k,n)的返回值。函数 ctr 根据 k 确定它在矩阵中的行号 i 和列号 j（注意行号和列号均从 0 算起），然后按矩阵存放原则计算出它在 b 中的位置。

[程序]

```
trans(b,n)
int n,b[];
{
int m,k,r,cc,rr;
int w;
m=(n+1)*n/2-4;
k=2;
while(m>0)
{
r=ctr(k,n);
if(r==k)
m--;
else
{
cc=k;rr=r;
while( (1) )
{cc=rr;rr=ctr(cc,n);}
if ( (2) )
{
cc=k;rr=r;w=b[k];
while( (3) )
{
b[cc]=b[rr];m--;
cc=rr;rr=ctr(cc,n);
}
b[cc]=w; (4) ;
}
}
k++;
}
}
ctr(k,n)
int k,n;
{
int i,j;
i=k;j=0;
while( (5) )
i-=++j;
return (i*n+j-i*(i+1)/2);
}
```

[解析]

由程序说明可知，从初始按行存放的数组转换为按列存放的数组－－不妨设为bb[ ]的过程中，不能使用内存数组，而是使用循环传送链。一旦一条传送链求出，这条传送链上的元素便可依次调整成功，以后再遇上这条链上其他元素时便无需再调整。

由例a[5][5]可知，该矩阵只有一条传送链b(2) →b(3) →b(6) →b(4) →b(10) →b(8) →b(11) →b(9) →b(5) →b(2)，另外b(7)=8未动。

程序中的值是可能需要调整的元素个数（数组b[ ]中的前两个和末两个不需要调整）。

1．子函数ctr(k,n)的功能：先求出bb[k]（按列存放）在a[ ][ ]中的行号i、列号j，然后，按照i、j求出a[i][j]在b[ ]（按行存放）中的位置r，并将r作为返回值，r=i*n+j-i*(i+1)/2。"while((5))"是求i、j的值。在此循环中，从i=k,j=0开始，每次j+1→j ，然后i-j→i，故若i≤j，则不再循环，（5）处应填"i>j"。

2．将b[r]放入bb[k]。

若r=k则不必再处理；否则需求出以k为首的传送链。初始k=2时，以bb[2]为首的链中的其他元素bb[p](p>2)均有p>2。由此可知，设当前位置为k，则若在以k为首的链中出现小于k的位置的元素时，说明链上的元素已调整，从而无需再调了。

"while((1))"循环便是求以k为首的传送链。故(1)处应填入"rr!=k && cc>=k"。rr!=k即此链还未求完（首尾未相接）；cc>=k即当前位置大于k。"if((2))"之后则可依次传送，故（2）处应为"cc>=k"，表明当前链上的元素未调整过。"while((3))"循环便是传送了，故（3）处应为"rr!=k"，直至链尾遇到链首（链首位置为k）。填空（4）处之前的"b[cc]=w(w=b[k]"是将链首位置的值放入链尾。既然此处又调整了一个元素，则（4）外显然应为"m--"。

[答案]

（1）rr!=k&&cc>=k

（2）cc>=k

（3）rr!=k

（4）m--

（5）i>j

历年试卷及答案（整套）

搞级程序员计算机软件专业技术资格和水平考试

1996 年试卷及答案

（1996 年度——上午试题）

试题 1o:p>

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

一棵二叉排序树可顺序存放在一组物理上相邻的存储区中，每个节点及其左、右指针依次存放在该存储区的三个连续单元中。现对一棵按节点字母的字典顺序构成的二叉排序树，从根节点 P 开始顺序存放在一个存储区中，结果如图 8-1 所示，其中 Li 为第 i 个结点的左指针，Ri 为第 i 个结点的右指针，则 L_2 应为_A_，L_4 应为_B_，R_1 应为_C_。该二叉排序树的前序遍历序列为_D_，后序遍历序列为_E_。

1000	P
1001	L_1
1002	R_1
1003	B
1004	L_2
1005	R_2
1006	Q
1007	L_3
1008	R_3
1009	H
100A	L_4
100B	R_4
100C	C
100D	L_5
100E	R_5
100F	J
1010	L_6
1011	R_6

供选择的答案：

A～C： ①1003 ②1004

③100A ④1009

⑤1006 ⑥1000

⑦100C ⑧100F

⑨Null

D、E： ①PBQHCJ ②PBHCJQ

③BCHJPQ ④CJHBQP

⑤BHCJQP

试题 2

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

在软件工程的设计阶段中，有三种常用的设计方法：结构化设计（SD）方法、Jackson方法和Parnas方法。SD方法侧重于_A_；Jackson方法则是_B_；Parnas方法的主要思想_C_。从70年代中期到90年代早期，_D_是最为常用的设计方法。_E_法只提供了重要的设计准则，没有规定出具体的工作步骤。

供选择的答案：

A～C： ①使用对象、类和继承 ②由数据结构导出模块结构

③模块要相对独立，且功能单一，使块间联系弱，块内联系强

④将可能引起变化的因素隐藏在有关模块内部，使这些因素变化时的影响范围受到限制

⑤用数据流图表示系统的分解，且用数据词典和小说明分别表示数据和加工的含义

⑥自顶向下、逐步细化，采用顺序、选择和循环三种基本结构，以及限制goto语句的使用，设计出可靠的和易维护的软件。

D： ①SD ②Jackson ③Parnas ④面向对象

E： ①SD ②Jackson ③Parnas ④以上皆非

试题3

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

设有一图书管理数据库，其关系模式是R_0（L#，B#，BNAME，BPRICE，BPUB），其属性分别表示个人借书证号、书号、书名、书价、图书出版社。该关系模式_A_。它的主要问题是数据冗余。如把R_0分解成两个关系模式R_1_B_和R_2_C_，则可以部分地解决这一问题。R_1和R_2是规范化程度较差的范式_D_。另外一种分解方法可以得到三个模式R_3（L#，B#），R_4（B#，BNAME），R_5（BNAME，BPRICE，BPUB），则R_3、R_4、R_5都_E_。

供选择的答案：

A、D、E： ①属于第一范式但不属于第二范式 ②属于第二范式但不属于

第三范式

③属于第三范式 ④不是范式

⑤属于第二范式但不属于第一范式 ⑥属于第三范式但不属于第二范式

B、C： ①（L#，B#，BPRICE） ②（L#，B#）

③（B#，BNAME） ④（B#，BNAME，BPRICE，BPUB）

⑤（BNAME，BPRICE，BPUB） ⑥（L#，BNAME，BPRICE）

试题 4

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

Internet 提供的服务有通信、远程登录、浏览、检索等。_A_直接用于人际通信，_B_用于远程登陆。_C_不是浏览软件。在浏览软件中，_D_不支持 HTML，_E_是目前微机上最常使用的浏览器。

供选择的答案：

A、B： ①WWW ②E-mail ③URL ④TCP/IP

⑤Telnet ⑥Lycos ⑦HTML ⑧PPP

C～E： ①Hotjava ②Netsacpe ③Mosaic ④Lycos

⑤Gopher

试题 5

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

有限状态自动机可用五元组（V_T，Q，δ，q_0，Qr）来描述，它可对应于_A_。设有一有限状态自动机 M 的定义如下： V_T ={ 0 ， 1 } Q={ q_0，q_1，q_2}

δ 定义为：

δ（q_0，0）=q_1　　δ（q_1，0）=q_2　　δ（q_2，1）=q_2　　δ（q_2，0）=q_2

Qr={q_2}。

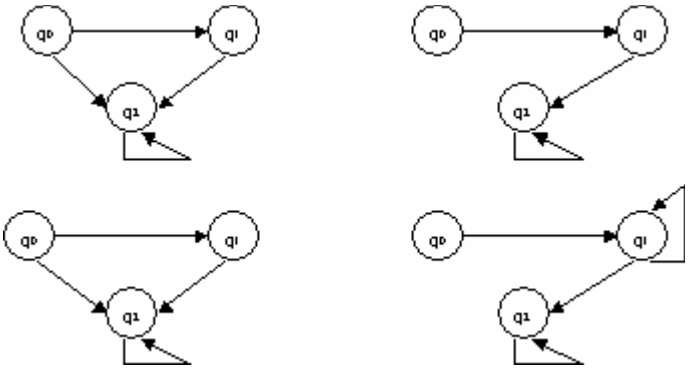
M 是一个_B_有限状态自动机，它所对应的状态转换图为_C_，它所能接受的语言可以用正则表达式表示为_D_，其含义为_E_。

供选择的答案：

A：①0 型文法　　②1 型文法　　③2 型文法　　④3 型文法

B：①歧义的　　②非歧义的　　③确定的　　④非确定的

注：图中-表示开始状态，+表示终止状态。



D：①$(011)^*$　　②$00(011)^*$　　③$(011)^*00$　　④$0(011)^*0$

E：①由 0 和 1 所组成的符号串的集合

②以为 0 头符号和尾符号、由 0 和 1 所组成的符号串的集合

③以两个 0 为结束的，由 0 和 1 所组成的符号串的集合

④以两个 0 为开始的，由 0 和 1 所组成的符号串的集合

试题 6

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

由于内存大小有限，为使得一个或多个作业能在系统中运行，常需要外存来换取内存。其中以作业为单位对内外存进行交换的技术称为_A_技术，在作业内部对内外存进行交换的技术称为_B_技术。用外存换内存是以牺牲程序运行时间为代价的。为提高 CPU 有效利用率，避免内外存的频繁交换，_B_技术常用某种_C_来选择换出内存的页面，它的基础是程

序的_D_。据此，操作系统可根据_E_来改善系统的性能，_E_是一个进程在定长的执行时间区间内涉及到的页面的集合。

供选择的答案：

A、B： ①SPOOLING ②SWAPPING ③虚拟存储 ④虚拟机

⑤进程管理 ⑥设备管理

C： ①页面分配策略 ②页面标志策略 ③页面淘汰策略 ④段设置策略

D： ①完整性 ②局部性 ③递归性 ④正确性

E： ①工作集 ②页面集 ③段号集 ④元素集

试题 7

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

软件测试的目的是_A_。通常_B_是在代码编写阶段可进行的测试，它是整个测试工作的基础。

逻辑覆盖标准主要用于_C_。它主要包括条件覆盖、条件组合（多重条件）覆盖、判定覆盖、条件及判定覆盖、语句覆盖、路径覆盖等几种，其中除路径覆盖外最弱的覆盖标准是_D_，最强的覆盖标准是_E_。

供选择的答案：

A： ①表明软件的正确性 ②评价软件质量 ③尽可能发现软件中错误 ④判定软件是否合格

B： ①系统测试 ②安装测试 ③验收测试 ④单元测试

C： ①黑盒测试方法 ②白盒测试方法 ③灰盒测试方法 ④软件验证方法

D、E： ①条件覆盖 ②条件组合覆盖 ③判定覆盖 ④条件及判定覆盖

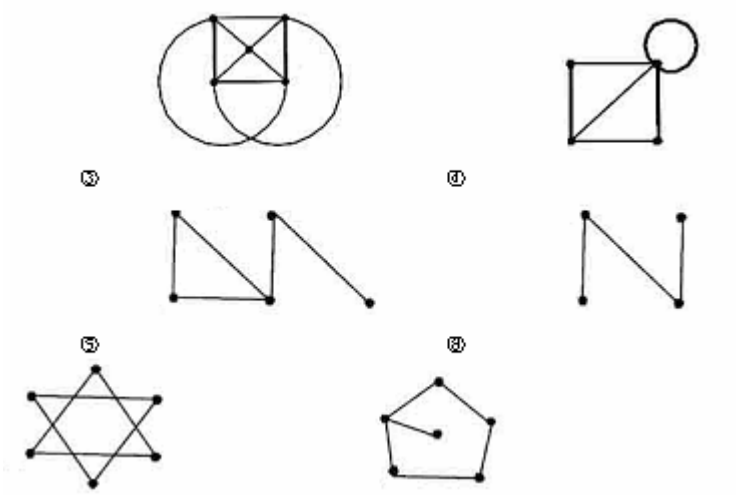
⑤语句覆盖

试题 8

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

下列图中，_A_是非简单图，_B_是完全图，_C_和_D_都是哈密尔顿图，其中_C_又是欧拉图，_E_是树。

供选择的答案：



试题 9

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

在多级存储系统中，Cache 处在 CPU 和主存之间，解决_A_问题。若 Cache 和主存的存取时间分别为 T_1 和 T_2，Cache 的命中率为 H，则该计算机实际存取时间为_B_，当 CPU 向存储器执行读操作时，首先访问 Cache ，如命中，则从 Cache 中取出指令或数据，否则从主存中取出，送_C_；当 CPU 向存储器执行写操作时，为了使 Cache 内容和主存的内容保持一致，若采用_D_法，同时写入 Cache 和主存。由于 Cache 容量比主存容量小，当 Cache 满时，但要执行把主存信息向 Cache 写入时，就要淘汰 Cache 中已有的信息，为了提高 Cache 的命中率，采用一种_E_替换算法。

供选择的答案：

A： ①主存容量扩充 ②主存和 CPU 速度匹配 ③多个请求源访问主存 ④BIOS 存放

B： ①H T_1+ T_2 ②(1-H T_1)+H T_2 ③T_2-H T_1 ④ H T_1+(1-H) T_2

C： ①Cache ②CPU ③Cache 和 CPU ④ Cache 或 CPU

D： ①写回 ②写通 ③映照 ④特征

E： ①LRU ②FIFO ③FILO ④RANDOM

试题 10

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

设图 1 和图 2 系统中，R_1、R_2、R_3 为三个加工部件，每个加工部件的失效率均为 λ，可靠性均为 R。则图 1 系统中的失效率为 _A_，可靠性为_B_；图 2 系统中的失效率为 _C_，可靠性为_D_。若每个加工部件的平均无故障时间为 5000 小时，则图 2 系统的平均无故障时间为_E_小时。

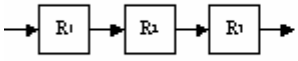


图 1

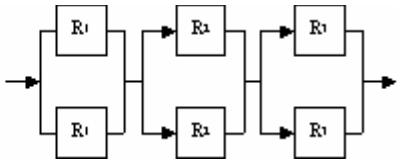


图 2

供选择的答案：

A： ①
v:shape id=_x0000_i1031 style="HEIGHT: 30.75pt; WIDTH: 12.75pt"
②
③
type="#_x0000_t75" o:ole="">
④

B： ① ② ③ ④

C： ① ② ③ ④

D： ①$(1-R^2)^3$ ②$3(1-R^2)$ ③$R^3(2-R)^3$ ④$1-3(1-R^2)$

E： ①2500 ②5000 ③7500 ④3333

试题 11

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

多媒体技术的关键在于解决动态图像和声音的存储与传输问题。若不经压缩，以VGA640*480 点阵存储一幅 256 色的彩色图像大约需_A_M 字节存储空间。以 9600bps 的速度传输这幅图像大约需_B_秒。按我国电视 PAL 标准每秒 25 幅，一张 650MB 的光盘可容纳约_C_秒这样的图像画面，播放时传送速率应不低于每秒_D_M 字节。模拟声音数字化存放是通过采样和量化实现的，若采样频率为 44.1KHz，每样本 16 位，存放一分钟双声道的声音约占_E_M 字节存储空间。

供选择的答案：

A～E： ①0.3 ②1.4 ③2.4 ④7.5

⑤10 ⑥32 ⑦78.6 ⑧87

⑨98.4 ⑩256

试题 12

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

数据加密是一种保证数据安全的方法，数据解密则是逆变换，即_A_。密码体制可分为_B_和_C_两大类。例如常用的 DES 属于_B_，而 RSA 则属于_C_。DES 的密钥长度为_D_位。

破密者面临多种不同的问题，其从易到难排列依次为_E_。

供选择的答案：

A： ①由加密密钥求出解密密钥 ②由密文求出明文

③由明文求出密文 ④由解密密钥求出加密密钥

B、C： ①公开密钥 ②替代密码 ③换位密码 ④对称密钥

D： ①32 ②48 ③64

④128

E： ①选择明文、已知明文、仅知密文 ②已知明文、仅知密文、选择明文

③已知明文、选择明文、仅知密文 ④仅知密文、已知明文、选择明文

试题 13

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

_A_ whether to go with a shared or switched fast LAN technology once you start to max out your shared Ethernet of token-ring LAN is tough call. Users and analysts involved in _B_ high-speed LANs have learned the benefits of each. However there are trade-offs to each, and _C_ which fast LAN technology to choose is critical for _D_ network performance problems and _E_ costly purchasing mistakes.

供选择的答案：

A～E： ①avoiding ②clipping ③choosing ④deciding

⑤dissolving ⑥ensuring ⑦evolving ⑧implementing

⑨knowing ⑩solving

试题 14

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

The most accurate and most boring way to _A_ Java is that it is a new computer programming language developed by Sun Microsystems that creates _B_ independent programs that can be distributed and run remotely. To run Java programs, a computer must have a Java interpreter. Currently _C_ Java programs are small “applets” that are _C_ as part of web pages.

When you use a Java _D_ browser to view a Web page that includes a Java applet, the browser loads the applet onto your computer through your modem or network. Then the Java interpreter runs the applet, which could include animation or sound, on your computer rather than

transmitting the code bit by bit over Internet. A few thousand bytes of Java code can turn into a powerful program on your computer.

So an applet could include _E_ with Java interpreter.

供选择的答案：

A： ①command ②comment ③describe ④discover

B： ①condition ②platform ③programmer ④ workstation

C、D： ①available ②capable ③possible ④ probable

⑤stable ⑥valuable

E： ①animation or sound ②animation and sound

③animation and be run ④animation and be loaded

试题 15

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

牛顿法和弦截法是求方程 f(x)=0 的根的两种常用方法。牛顿法对于单根_A_，对于重根_B_，弦截法_C_，弦截法是在根 x 临近用_D_作为 f(x)的近似式。如果保留前一函数值而不重新计算，一般来说，要达到指定精度，与弦截法比较，牛顿法_E_。

供选择的答案：

A～C： ①1 阶收敛 ②2 阶收敛 ③ 3 阶收敛 ④ 可 能不收敛

⑤肯定不收敛 ⑥一致收敛

D： ①常数 ②三角函数 ③f(x)的一次插值函数 ④f(x)的二次插值函数

E： ①迭代次数少，每次迭代的计算量少 ②迭代次数少，每次迭代的计算量多

③迭代次数多，每次迭代的计算量少 ④迭代次数多，每次迭代的计算量多

（1996 年度——下午试题）

试题一

阅读下列说明和流程图，回答问题 1 至问题 2，把解答写在答卷的对应栏内。

【说明】

本流程图将数字 1，2，…，N^2（N≥2）按逆时针方向依次写在 N*N 矩阵中，下图给出了 N=4 和 N=5 时的情况：

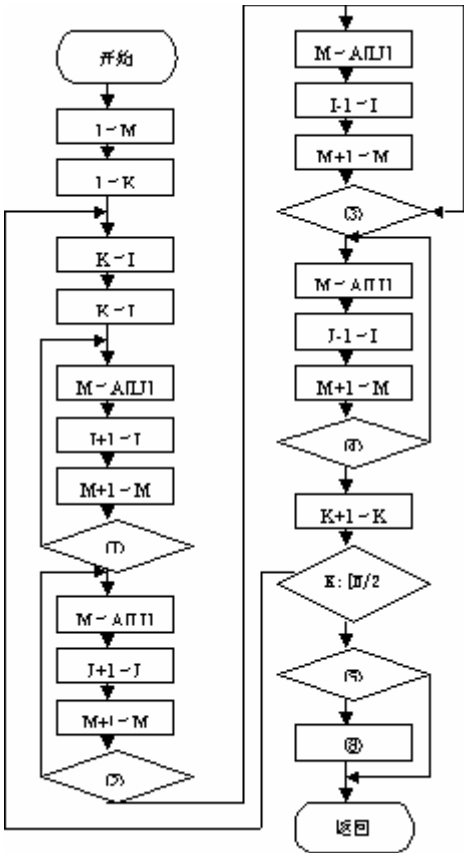
1	12	11	10
2	13	16	9
3	14	15	8
4	5	6	7

N=4 时

1	16	15	14	13
2	17	24	23	12
3	18	25	22	11
4	19	20	21	10
5	6	7	8	9

N=5 时

【流程图】



注：图中[W]表示不大于 W 的最大整数。

［问题 1 ］

填充流程图中的①～⑥使之成为完整的流程图。

［问题 2 ］

若将数字 1，2，…，N^2 按顺时针方向依次写在 N*N 矩阵中，则只需将上述流程图中的_⑦_改成_⑧_即可。

试题二

阅读下列说明和流程图，回答问题 1 至问题 3 ，把解答写在答卷的对应栏内。

【说明】

本流程图描述了某行业分类电话号码簿（简称号簿）出版系统的处理流程。全市所有电话的基本信息均存放在营业库中。系统输入工单，工单中包括电话的新装、拆除、移机、更

改（更改户名、地址、电话号码等）等信息。为确保输入工单的正确性，每张工单均由两个录入员分别录入，由处理 1 进行输入和校对，然后更新营业库。系统根据待出版号簿的行业类型从营业库中选取该类用户的电话信息，存放在号簿中。同时向每个电话用户发布用户函，用户函上记录着将刊登在号簿上的该用户的户名、地址、电话号码等信息，用户函上的序号标志着该用户信息在号簿库中的位置。用户受到用户函后自行校对，并将修改内容和印刷要求（字体大小和是否套红）填写在用户回执中，系统按收到用户回函的先后顺序依次输入用户会函，然后更新号簿库。最后通过排版输出经用户校对并符合其印刷要求的号簿清样。

系统中部分单据和文件的格式如下：

工单＝工单类型＋原户名＋新户名＋原地址＋新地址＋原电话号码＋新电话号码

营业库纪录＝户名＋地址＋电话号码＋分类信息

用户函＝序号＋户名＋地址＋电话号码

用户回函＝序号＋户名＋地址＋电话号码＋套红标记＋字体大小

【流程图】

［问题 1 ］

流程图中哪些处理能发现工单的哪些错误，并举例说明。

［问题 2 ］

指出号簿库文件的纪录至少应包括哪些数据项。

［问题 3］

为提高处理速度，流程图需作何改进。

试题三

阅读以下说明和 E-R 图，回答问题，讲解答写在答卷的对应栏内。

【说明】

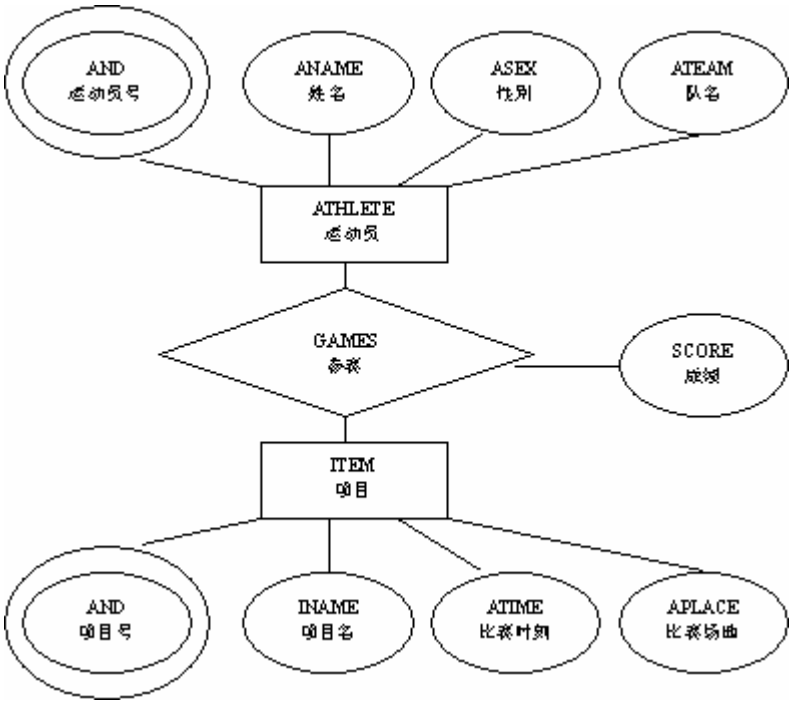
设有下列关于运动会管理系统的 E-R 图。图中矩形表示实体，圆表示属性，双圆表示关键字属性，菱形表示实体之间的关系。假定已通过下列 SQL 语言建立了基本表：

CREATE TABLE ATHLETE

```
(ANO CHAR(6) NOT NULL,
 ANAME CHAR(20),
ASEX CHAR(1),
ATEAM CHAR(20));
CREATE TABLE ITEM
(INO CHAR(6) NOT NULL,
 INAME CHAR(20),
ISEX CHAR(1),
ITEAM CHAR(20));
CREATE TABLE GAMES
(ANO CHAR(6) NOT NULL,
 INO CHAR(6) NOT NULL,
SCORRE CHAR(10));
```

为了答题的方便，图中的实体和属性同时给出了中英文两种名字，回答问题时只需写出英文名即可。

【E-R 图】



［问题］

填充下列 SQL 程序 3.1～3.4 中的①～⑦，使它们分别完成相应的功能：

程序 3.1：统计参加比赛时运动员人数

SELECT _①_

FROM ATHLETE

WHERE ASEX='M'；

程序 3.2：查 100872 号运动员参加的所有项目及其比赛时间和地点

SELECT ITEM,INO,INAME,ITIME,IPLACE

FROM GAMES,ITEM

WHERE _②_

AND _③_;

程序 3.3：查参加 100035 项目的所有运动员名单

SELSECT ANO,ANAME,ATEAM

FROM ATHLETE

WHERE _④_

(SELECT_⑤_

FROM GAMES

WHERE GAMES.ANO=ATHLETE.ANO AND INO=’100035’);

程序 3.4：建立运动员成绩视图

_⑥_ ATHLETE_SCORE

AS SELECT ATHLETE,ANO,ANAME,ATEAM,INAME,SCORE

FORM _⑦_

WHERE ATHLETE,ANO=GAMES,ANO AND GAMES.INO=ITEM.INO;

试题四为必答题。

试题四

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读下列程序说明和 CASL 程序，将应填入程序中_?_处的字句，写在答卷的对应栏内。

【程序说明】

子程序 OFFSET 用二分法，查找无符号整数 M 在一个长度为 N 的有序（升序）无符号整数列表 NTABLE 中的位置。

程序中标号为 LOW 和 UP 的两个存储字分别用作存放查找区间的上下限。

进入子程序时，在 GR1 中给出存放子程序所需参数的起始地址。参数的存放次序如下：

(GR1)+0	M
+1	N
+2	NTABLE 的首址

从子程序返回时，GR0 中存放查找结果，即 M 在此有序表中的位置序数。如表中找不到 M，则 GR0 中返回 0。其它寄存器的内容保持不变。

【程序】

```
        START
OFFSET  PUSH    0,GR2
        PUSH    0,GR3
        LD      GR0,0,GR1
        LEA     GR2,0
        ST      GR2,LOW
        _①_
        _②_
        ST      GR2,UP
LOOP    ADD     GR2,LOW
        SRL     GR2,1
        LEA     GR3,0,GR2
        _③_
        _④_
        JZE     FOUND
        JPZ     INCLOW
        LEA     GR2,-1,GR2
        ST      GR2,UP
        JMP     CMPLU
INCLOW  LEA     GR2,1,GR2
        ST      GR2,LOW
        _⑤_
CMPLU   CPL     GR2,LOW
```

```
        _⑥_
        _⑦_
FOUND   LEA     GR0,1,GR2
        POP     GR3
        POP     GR2
        RET
LOW     DS      1
UP      DS      1
        END
```

从下列的 4 道试题（试题五至试题八）中任选 2 道解答。如果解答的试题数超过 2 道，则解答的前 2 道有效。

试题五

阅读以下程序说明和 C 程序，将应填入程序中_?_处的字句，写在答卷的对应栏内。

【程序说明】

本程序是一个简单的计算器模拟程序。对任意给定的正确四则运算表达式，程序计算其结果值并输出。表达式中运算分量为无正负号整数，运算符为+、_、*、/，圆括号按常规配对，表达式以字符“=”结束。

函数 getach()为获取表达式的一个合法自负，并将字符存入变量 curch；函数指针树组 func[]是为了统一加减乘除计算而设置的。

【程序】

```
#include  <stdio.h>
int   add(int x,int y){return x+y;}
int   sub(int x,int y){return x-y;}
```

```
int    mul(int x,int y){return x*y;}

int    div(int x,int y){return x/y;}

int    (*func[])()={add,sub,mul,div};

int    num,curch;

char chtbl[]="+-*/()=";

char corch[]="+-*/()=0123456789";

int    getach()

{      int   i;

              while(1)

              {      curch=getchar();

                     if(curch==EOF) return –1;

                     for(i=0;corch[i]&curch!=corch[i];i++);

                     if(i<strlen(corch)) break;

              }

              return      curch;

}

int   geid()

{      int   I;

              if(curch>='0'&&curch<='9')

              {      for(num=0;curch>='0'&&curch<='9';getach())

                     num=_①_;

                     return      –1;

              }

              else{for(i=0;chtbl[i];i++)
```

```
                if(chtbl[i]==curch) break;
                    if(i<=5) getach();
            return      i;
        }
    }

int   cal()
{     int   x1,x2,x3,op1,op2,i;
            i=getid();
            if(i==4) x1=cal();    else x1=num;
            op1=getid();
            if(op1>=5) return x1;
            i=getid();
            if(I==4) x2=cal();    else x2=num;
            op2=getid();
            while(_②_)
            {     i=getid();
                  if(i==4) x3=cal();    else x3=num;
                  if((op1/2==0)&&(op2/2==1))
                        x2=(*func[op2](x2,x3));
                  else{       x1=_③_;
                        x2=x3;
                        _④_;
                  }
```

```
            op2=getid();
        }
            return    _⑤_(x1,x2);
    }
    void main()
    {   int  value;
            printf("Please input an expression:\n");
            getch();
            while(curch!='=')
            {   value=cal();
                printf("The result is : %d\n",value);
                printf("Please input an expression:\n");
                getach();
            }
    }
```

试题六

阅读下列程序说明和 FORTRAN 程序，将应填入程序中_?_处的字句，写在答卷纸的对应栏内。

【程序说明】

子程序 SUM 计算数列

1，1/2，1/3，…，1/n，…

的前 N 项和，并以 M 位小数形式输出（M≤60）。

为提高计算结果的精度，用分数形式计算并存放数列的部分和，求和结果记为

A+U/V

其中 U/V 是不可约真分数，A 为整数。

例如：n=5，M=10 时，子程序输出为

1+1/2+1/3+1/4+1/5=2+17/60

2.283333333

整型函数 GCD 用辗转相除法计算 U 和 V 的最大公约数。

【程序】

```
      SUBROUTINE      SUM(N,M)
      INTEGER         A,U,V,G,D(60),GCD
      A=1
      U=0
      V=1
      DO  20   K=2,N
        U=_①_
        V=_②_
        A=_③_
        U=MOD(U,V)
        _④_
        U=U/G
        V=V/G
20    CONTINUE
      WRITE(*,100)N,A,U,V
```

```
100  FORMAT(1X,'1+1/2+1/3+…+1/',I2,'=',I2,'+'I10,'/',I10)
     DO  40   I=1,M
          D(I)=_⑤_
          _⑥_
40   CONTINUE
     WRITE(*,200)A,(D(I),I=1,M)
200  FORMAT(1X,I2,'.',60I1)
     END
     INTEGER      FUNCTION   GCD(U,V)
     INTEGER                 U,V
     K=V
     L=U
10   IF(MOD(K,L).GT.0) THEN
          J=MOD(K,L)
          K=L
          L=J
          GOTO    10
     ENDIF
     _⑦_
     END
```

试题七

阅读下列程序说明和 C 程序，将应填入程序中_?_处的字句，写在答卷纸的对应栏内。

【程序说明】

本程序先从文件读入各考生的准考证号（设为整型数）及成绩，并将其存放在一棵检索二叉树上，二叉树的结点的键值是成绩，二叉树每个节点带一链表，链表结点存放取得该成绩的考生的准考证号。然后，程序按中序遍历检索二叉树，从高到底分输出结果，使每行输出某成绩及其取得该成绩的各考生的准考证号。

【程序】

```
#include  <stdio.h>
typedef struct idnode{
        int             id;
        struct idnode   *next;
}IdNode;
typedef struct marknode{
        int             mark;
        IdNode          *head;
        struct marknode     left,right;
}MarkNode;
char  fname[]=”sp07.dat”’
main()
{       int             id,mark;
        MarkNode            *root=null;
        FILE                *fp=fopen(fname,”r”);
        if(!fp){
        printf(“file %s open error.\n”,fname);
         exit(0);
        }
        while(!feof(fp)){
```

```
            fscanf(fp,"%d%d",&id,&mark);
            btree(&root,id,mark);
        }
        fclose(fp);
        print(root);
    }
    btree()
    {   IdNode          *ip;
        MarkNode            *mp=mpptr;;
        if(_①_){
            if(mark==mp->mark) addIdNode(_②_,id);
            else  if(mark>mp->mark) btree(&mp->left,id,mark);
                else btree(&mp->right,id,mark);
        }else
        {   mp=(MarkNode *)malloc(sizeof(MarkNode));
            mp->mark=mark;
            mp->left=mp->right=NULL;
            _③_
            addIdNode(&mp->head,id);
            _④_
        }
    }
    addIdNode(IdNode **ipp,int id)
    {   IdNode    *ip=ipp;
```

```
        if(_⑤_) addIdNode(_⑥_,id);
        else{
            ip=(IdNode *)malloc(sizeof(IdNode));
            sp->id=id;
            ip->next=NULL;
            _⑦_
        }
}
print(MarkNode *mp)
{   IdNode *ip,ip0;
    if(mp){
    print(mp->left);
    printf("%6d: \t",mp->mark);
    ip=mp->head;
    while(ip){
        printf("%6d: \t",ip->id);
        ip0=ip;
        ip=ip->next;
        free(ip0);
    }
    printf("\n");
    print(mp->right);
    free(mp);
}
```

试题八

阅读下列程序说明和 FORTRAN 程序，将应填入程序中_?_处的字句，写在答卷纸的对应栏内。

【程序说明】

某公司招聘 M 个工种（编号为 1～M）的工作人员，每个工种有各自的计划招工数。共有 N 位应聘者，每位应聘者有一报名号，且必须申报两个工种志愿，并参加公司组织的笔试和面试。公司为每位应聘者评定一个综合考试成绩（0～100 分）。然后从高分到低分（分数相同者报名号小的优先）依次对每个应聘者进行录用。录用的原则如下：

（1）对同一应聘者，第一志愿优于第二志愿。

（2）若应聘者的第一志愿工种已录满，则将其成绩减去 5 分后，立即参加第二志愿录用。

（3）在一个工种录取时，按申报该工种的应聘者“录用成绩”（第一志愿为考试成绩，第二志愿为考试成绩减 5 分）从高到底次序录用，“录用成绩”相同者，按报名号从小到大顺序优先录用。

（4）允许某工种未招满计划招工数。

程序中数组 NO、MARK、ZY1、ZY2 分别存放应聘者的报名号、考试成绩、第一志愿工种遍号和第二志愿工种编号，并假定它们已先按考试成绩降序、后按报名号升序的顺序排列，数组 GZ 存放各工种的计划招工数。

子程序 RY 按上述录用原则完成录用工作。录用过程中使用了三维数组 RYNO，数组的最终值即为各工种的录取顺序名单。数组元素 RYNO(K,J,1)和 RYNO(K,J,2)分别存放第 K 个工种准备录用的第 1 名应聘者的报名号和“录用成绩”。数组元素 TOP(K)存放录用过程中第 K 个工种已存入数组 RYNO 的人数。

子程序 INSERT 将报名号 NOI 和成绩 MARKI 按“录用成绩”降序顺序插入到数组 RYNO 第 K 个工种的相应位置中。

【程序】

```
SUBROUTINE      RY(NO,MARK,ZY1,ZY2,GZ,RYNO,TOP,M,N)
INTEGER         NO(N),MARK(N),ZY1(N),ZY2(N),GZ(N)
INTEGER         RYNO(M,N,2),TOP(,M)
DO  10   I=_①_
```

```
10        TOP(I)=0
      DO  20   I=1,N
          K1=ZY1(I)
          K2=ZY2(I)
          IF(TOP(K1).LT.GZ(K1))      THEN
              _②_
              CALL     INSERT(RYNO,M,N,K1,TOP(K1),NO(I),MARK(I))
          ELSEIF(MARK(I).GT.RYNO(K1,TOP(K1),2))   THEN
                  CALL     INSERT(RYNO,M,N,K1,TOP(K1),NO(I),MARK(I))
              ELSEIF(TOP(K20.LT.GZ(K2))     THEN
                  TOP(K2)= _③_
                  RYNO(K2,TOP(K2),1)=NO(I)
                  RYNO(K2,TOP(K2),2)=_④_
          ENDIF
20    CONTINUE
      END
      SUBROUTINE          INSERT(RYNO,M,N,K,TOPK,NOI,MARKI)
      INTEGER             RYNO(M,N,2),TOPK
10    IF(J.EQ.0)     THEN
          RYNO(K,1,1)=NOI
          RYNO(K,1,2)=MARKI
      ELSEIF(MARKI_⑤_RYNO(K,J,2))    THEN
              RYNO(K,J+1,1)=NOI
              RYNO(K,J+1,2)=MARKI
```

```
        ELSE
            _⑥_
            _⑦_
            J=J-1
            GOTO    10
    ENDIF
    END
```

上午试题答案

	A	B	C	D	E
1	9	7	5	2	4
2	3	2	4	1	3
3	1	2*	4*	2	3
4	2	5	4	5	2
5	4	3	/	/	/
6	2	3	3	2	1
7	3	4	2	5	2
8	2	4	3	2	1
9	3	2	4	3	1
10	1	10	8	4	5
11	2	4	1	3	1
12	4	8	9	10	1
13	3	2	1	2	3
14	2	1	1	2	4
15	2	5	4	3	2

注：*表示可交换 因试题 5 中 c 的供选择答案有印刷错，故 C、D、E 的任何解答均作正确处理

下午试题答案

- **流程图**

试题一

［问题 1］

处理 1 能发现两人录入不全同的工单和非法数据（如非法字符，数据越界等）

处理 2 能发现与营业库不一致的工单（如新装电话重号，移机，拆机，更改等原电话号码不存在等）

处理 4 能发现两人录入不全同的用户回函和非法数据(如非法字符,电话号长度不对等)

［问题 2］

户名+地址+电话号码+套红标记+字体大小（"套红标记+字体大小"可写成"印刷要求"）

［问题 3］

处理 5 前对回函库按序号排序。

处理 2 前对工单库的排序关键字进行排序

试题二

［问题 1 ］

1. I:N-K 或 I;N-J 或 M:4(K-1)(N-K+1)+N-2K+1

2. J:N-K 或 J:N-J 或 M:4(K-1)(N-K+1)+2(N-2K+1)

3. I:K+1 或 I;N+2-J 或 4(K-1)(N-K+1)+3(N-2K+1):M

4. I:K+1 或 J;I+1 或 4K(N-K):M

5. MOD(N,2):0 或 [N/2]*2:N 或 [N/2]:[(N+1)/2] 或 N/2:[N/2] 或 [N/2]:[N/2] 或 N:2(K-1) 或 I+J=N 或 M-1:N*N

6. M→A[K,K] 其中：M 可答成 N*N; A[K]可答成[(N+1)/2, (N+1)/2]或 A[[N/2]+1, [N/2]+1] A[I+1,J+1]

［问题 2］

1. A[I,J] 或 M→A[I,J]

2. A[J,I] 或 M→A[J,I]

试题三

1. COUNT(*)

2. GAMES.INO=ITEM.INO

3. GAMES.ANO='100872' 注: 2,3 可互换

4. EXISTS

5. * 或 ANO 或 INO 或 SCORE 或三个列名的任意组合 4,5 也可为 4. ANO，IN 5. ANO

6. CREATE VIEW

7. ATHLETE, ITEM, GAMES(三项可交换)

- **汇编**

试题四

1. LD GR2,1,GR1 或 ADD GR2,1,GR1

2. LEA GR2,-1,GR2 1,2 两条指令中形成 N-1 的解答均正确

3. ADD GR3,2,GR1

4. CPL GR0,0,GR3

5. LD GR2,UP

6. JPZ LOOP

7. LEA GR2, -1 或 SUB GR2, LOW

- **C 语言**

试题五

1. num*10+curch-'0' 其中'0'可答成 48 或 0x30 或 corch[7]

2. (op2>=0)&&(op2<5) 或 op2<5 或 op2<=4 或 !(op2>=5)

3. (*func[op1](x1, x2)

4. op1=op2

5. (func[op1])

试题七

1. mp 或 mp !=NULL 或 *mpptr 或 *mpptr !=NULL\
2. &mp->head 或 &(mp->head)
3. mp->head=NULL
4. *mpptr=mp
5. ip 或 ip !=NULL 或 *ipp 或 *ipp !=NULL
6. &ip->next 或 &(ip->next)
7. *ipp=ip

- **FORTRAN**

试题六

1. U*K+V
2. V*K
3. A=U/V 或 A+INT (U/V) 或 A+(U-MOD(U,V))/V
4. G=GCD(U,V)
5. 10*U/V 或 INT(10*U/V) 或 (U*10-MOD(U*10,V))/V
6. U=10*U-D(I)*V 或 U=MOD(10*U, V)
7. GCD=L

试题八

1. 1,M 或 1,M,1
2. TOP(K1)=TOP(K1)+1
3. TOP(K2)+1
4. MARK(I)-5
5. .LE.

6. RYN0(K,J+1,1)=RYNO(K,J,1)

7. RYN0(K,J+1,2)=RYNO(K,J,2)

注：(6), (7)次序可交换

高级程序员计算机软件专业技术资格和水平考试

1997 年试题与答案

（1997 年度——上午试题）

试题 1

从供选择的答案中，选出应填入下面叙述中____?____内的最确切的解答，把相应编号写在答卷的对应栏内。

一咱最早用于科学计算的程序设计语言是__A__；一种提供指针和指外操作且不存在布尔类型蝗、应用广泛的

系统程序设计语言是__B__；一种适合在互联网上编写程序可供不同平台上运行的面向对象程序设计语言是__C__;

一种在解决人工智能总是上使用最多的有强的表处理功能的函数程序设计语言是__D__；一种以谓词逻辑为基础的，

核心是事实、规则和推理机制的实用逻辑程序设计语言是__E__。

供选择的答案

A~E： ① PASCAL ② ADA ③ SMALLTALK ④ SNOBOL⑤

C ⑥ ALGOL 68 ⑦ JAVA ⑧ LISP⑨ PROLOG ⑩ FORTRAN

试题 2

从供选择的答案中，选出应填入下面叙述中___?___内的最确切的解答，把相应编号写在答卷的对应栏内。

在有一台处理机 CPU 和两台输入输出设备 IO1 和 IO2，且能够实现抢先式多任务并行工作的多道程序内，投入运

行优先级由高到低 P1，P2，P3 三个作业 。它们使用设备的先后顺序和占用设备时间分别是：

作业 P1：I02（30 毫秒） CPU（10 毫秒） IO1（30 毫秒） CPU（10 毫秒）

作业 P2：IO1（20 毫秒） CPU（20 毫秒） IO2（40 毫秒）

作业 P3：CPU（30 毫秒） IO1（20 毫秒）

在对于其它辅助操作时间可以忽略不计的假设下，作业 p1,p2,p3 从投入到完成所用的时间分别是__A__毫秒，

__B__毫秒和___C___毫秒。三个作业从投入运行到全部完成，CPU 的利用率约为__D__%,IO1 的利用率约为

__E__%。

假定在系统中仅有这三个作业投入运行，各设备的利用率指该设备的使用时间同作业进程全部完成所占用最长

时间的比率。
供选择的答案
A~E： ① 60 ② 67 ③ 70 ④ 78⑤ 80 ⑥ 89 ⑦ 90 ⑧ 100 ⑨ 110 ⑩ 120

试题 3
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
当网络用户通过网络与另一台主机 X 通信，发现响应太慢时，可运行__A__程序，把一个分组发向主机 X，通过
查看所返回的分组首部的 __B__，发现问题的所在，并采取相应的措施。
对一个大中型网络，需要有一个网络管理系统进行管理。当前流行的各类网管平台软件都支持__C__协议。驻
留在__D__上的网管平台软件可通过该协议软件调阅被管理的网络结点内的管理信息库中的内容。有若干常用
的网络平台软件，但__E__不是网管平台软件。
供选择的答案
A：` ① Browser ② Bitware ③ Pint ④ Handshaking
B: ① 地址 ② 时戳 ③ 标识码 ④ 校验码
C： ① MAP ② SNMP ③ MHS ④ FTAM
D： ① 数据库服务器 ② 通信服务器 ③ 主路由器 ④ 网络管理工作站
E： ① NetManager ② OpenView ③ SunView ④ NetView

试题 4
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
设数据结构（D，R）由数据结点集合 D={di | 1<i<7 }及其上的关系 R 组成。
当 R={<di-1,di>|di-1,di D,2<i<7},这个数据结构对应于__A__。
当 R={<d4,d2>,<d2,d1 >,<d2,d3>,<d4 ,d6>,<d6,d5 >,<d6,d7>} 这个结构图形是__B__;用__C__遍历法可
以得到__A__的数据结构。当 R={<d1,d2 >,<d1,d3 >,<d2,d4>,<d3,d4>,<d4,d5>,<d4,d6>,<d4,d7>},
这个结构的图形是__D__;用__E__遍历法可以得到__A__的数据结构。
供选择的答案
A、B、D： ① 二叉树 ② 队列 ③ 二叉排序树 ④ 线性表 ⑤ 无向图 ⑥ 有向无回路图
C、E ： ① 前序 ② 中序 ③ 后序 ④ 深度优先 ⑤ 广度优先

试题 5
从以下叙述中选出 5 条最确切的叙述，把相应编号依次写在答卷的 A～E 栏内。
在数据库系统中，数据独立性指数据之间的相互独立，互不依赖。
SQL 语言的视图定义和视图操作功能不支持逻辑数据的独立性。
SQL 语言中不提供显式地使用索引的功能，支持了物理数据的独立性。
用户对“脏数据”的读出是由于数据库完整性规则受到了破坏。
在数据库系统中，数据的安全性是指保护数据以防止未被授权用户的蓄意或者无意使用。
实体完整性规则指主关键字值的任何组成部分都不可以是空值；引用完整性规则则不允许引

用不存在的实体
（即元组）。
在数据库系统中，数据的完整性是指数据的正确性和有效性。
“授权”是数据库系统中采用的完整性措施之一。
事务处理(Transaction)是数据库运行的基本单位。如果一个事务处理成功，则全部数据行到更新和提交；如
果失败，则已做的全部更新被恢复成原状，好象整个事务处理未进行过一样。这样使数据库保持了一致性。
对数据库的查找、增添、删除、修改等操作都需由数据库管理员进行完整性定义和完全性授权，由数据库系统
具体执行。

试题 6
从供选择的答案中，选出应填入下面叙述中 __?__ 内的最确切的解答，把相应编号写在答卷的对应栏内。
在设计测试用例时，__A__是用得最多的一种黑盒测试方法。在黑盒测试方法中，等价类划分方法设计测试用
例的步骤是：
根据输入条件把数目极多的输入数据划分成若干个有效等价类和若干个无效等价类；
设计一个测试用例，使其覆盖__B__尚未被覆盖的有效等价类，重复这一步，直至所有有效等价均被覆盖。
设计一个测试用便，使其覆盖__C__尚未被覆盖的无效等价类，重复这一步，直至所有无效等价均被覆盖。
因果图方法是根据__D__之间的因果关系来设计测试用例的。
在实际应用中，一旦纠正了程序中的错误后，还应选择部分或全部原先已测试过的测试用例，对修改后的程序
重新测试，这种测试称为 __E__。
供选择的答案
A： ① 等价类划分 ② 边值分析 ③ 因果图 ④ 判定表
B、C： ① 1WH ② 7 个左右 ③ 一半` ④ 尽可能少的 ⑤ 尽可能多的 ⑥ 全部
D： ① 输入与输出 ② 设计与实现 ③ 条件与结果 ④ 主程序与子程序
E： ① 验收测试 ② 强试测试 ③ 系统测试 ④ 回归测试

试题 7
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
用高级语言编写程序时，子程序调用语名中的实在参数必须与子程序说明中的形式参数在__A__ 上保持一致在
允许子程序递归调用的高级语言环境中，需用动态存储管理方法，它通常使用一个__B__存入子程序的调用记
录，调用记录可包括（1）全局量存储区域的__C__(2)调用点所在子程序的__D__；（3）调用点的__E__；
（4）形式参数和实在参数的通信区域；（5）返回值；（6）本子程序的局部量和临时变量存储区域等。

供选择的答案
A： ① 个数、类型 ② 个数、顺序 ③ 个数和、格式、顺序 ④ 个数、类型、顺序
B： ① 线性表 ② 队列 ③ 堆 ④ 下推栈
C～E： ① 子程序首地址 ② 调用记录首地直
③ 参数地址 ④ 寄存器地址
⑤ 返回地址 ⑥ 开始地址

试题 8
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
某 CPU 的主振频率为 100 兆赫，平均每个机器周期包含 4 个主振周期。各类指令的平均机器周期数和使用频度如
下表所示，则该计算机系统的速度为平均约__A__兆指令/秒。

指令类别

访内

一般算术逻辑运算

比较与转移等

乘除

其它

平均机器周期数/指令

2,5

1.25

1.5

15

5

使用频度

25%

40%

25%

5%

5%

若某项事务处理工作所要执行的机器指令数是：控制程序（以访内、比较与转移等其它指令为主）220000 条
指令和业务程序（以包括乖除在内的算术逻辑运算为主）90000 条指令，且指令使用频度基本如上表所示，则
该计算机系统的事务处理能力约为__B__项/秒。
若其它条件不变，仅提高主振频率至 150 兆赫，则此时该计算机速度为平均约__C__兆指令/秒，对上述事务的
处理能力约为__D__项/秒。
若主频仍为 100 兆赫，但由于采用了流水线和专用硬件等措施，使各类指令的每条指令平均机器周期数都变为
1.25，此时，计算机的速度平均约__E__ 兆指令/秒。
供选择的答案
A ～E： ① 1 ② 5 ③ 10 ④ 15 ⑤ 20
⑥ 33.3 ⑦ 50 ⑧ 66.7 ⑨ 100 ⑩ 200

试题 9
从选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
SCSI 是一种通用的系统级标准输入/输出接口,其口__A__标准的数据宽度 16 位，数据传送率达 20MB/S。
大容量的辅助存贮器常采用 RAID 磁盘阵列。RAID 的工业标准共有六级。其中__B__是镜象磁盘阵列，具有最高
的安全性；__C__是无独立校验盘的奇偶校验码磁盘阵列；__D__是采用纠错海明码的磁盘阵列；__E__则是既
无冗余也无校验的磁盘阵列，它采用了数据分块技术，具有最高的 I/O 性能和磁盘空间利用率，比较容易管理，
但没有容错能力。
供选择的答案
A： ① SCSI-I ② SCSI-II
③ FAST SCSI-II ④ FAST/WIDE SCSI-II
B～E： ① RAID 0 ② RAID 1 ③ RAID 2 ④ RAID 3
⑤ RAID 4 ⑥ RAID 5

试题 10
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
语言与文字输入技术是研究如何将计算机主要靠键盘输入文字数据的方式逐步

改变成人们口述或写入的方式送
入信息。语言与文字输入的过包含有__ A__、预处理、特征抽取、__B__与分类决策等环节。__A__阶段通过
传感器获得的模拟电信号要经过模数转换变成数字信号，它需要对模拟电信号抽样测量，将测量的值__C__，
以便用二进制数字信号来表示；预处理的主要任务是进行削弱无用信息和增强有用信息的工作；特征抽取将上
述环节产生的输入样本以有利于决策的形式表示出来，典型的常用方法是__D__表示法；__B__又称为分类器学
习；分类决策则通过比较和决策来完成对输入信息的 __E__。
从选择的答案
A～E： ① 辨认识别 ② 特征向量 ③ 频谱分析 ④ 分级取整 ⑤ 消除噪声 ⑥ 模型生成 ⑦ 数据合成 ⑧ 信息获取

试题 11
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
计算机的发展已进入了网络计算的新时代。Internet 是目前世界范围内最大的互联网。如此多的各种计算机
之所以能通过 Internet 相互通信，是因为它们遵循一了套共同的 Internet 协议。这套协议的核心是__A__，
在其上建立的无连接的运输层协议是__B__，万维网 WWW 上超文本传输遵循__C__，电子邮件传输遵循__D__，
Ethernet 与 Internet 连接时要用__E__。
供选择的答案
A～E： ① TCP ② IP ③ EDIFACT ④ HDLC
⑤ ARP ⑥ UDP ⑦ FTP ⑧ ICMP
⑨ SMTP ⑩ HTTP

试题 12
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
The relational database model requires the data be __A__ through programs that don’t
rely on the position of the data in the database. This is in direct __B__ to the other
database mode, where the program has to follow a series of pointers to the data it seeks.
A program __C__ a relational database simply asks for the data it seeks; the DBMS performs
the necessary searches and __D__ the information. The __E__ on how the search is done are
specific to the DBMS and vary from product to product.
供选择的答案

A： ① accessed ② moved ③ read ④ wrote
B: ① conduct ② contract ③ contrast ④ construct
C: ① consulting ② containing ③ querying ④ queuing
D: ① erases ② provides ③ proves ④ values
E: ① details ② documents ③ tails ④ tenants

试题 13
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
When most people refer to multimedia, they generally mean the combination of two or more
continuous media, usually with some user __A__. In practice, the two media are normally
audio and video , this is ,__B__ plus moving __C__.
It should be obvious by now that transmitting multimedia material in uncompressed form is
completely out of __D__. The only hope is that massive compression is possibe. Fortunately,
a large body of research over the past few decades has led to many compression thchniques
and algorithms that make multimedia transmission_ __E___.
供选择的答案
A～C： ① display ② games ③ help ④ interaction ⑤ pictures ⑥ sound ⑦ web
D、E: ① impossible ② fearful ③ feasible ④ program ⑤ question ⑥ ting

试题 14
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
包含 3 个命题变元的命题公式中，能够出现__A__个真值赋值。包含 3 个命题变元的命题公式可以形成__B__个
不同真值的命题公式。
公式__C__是予盾式，公式__D__是非重言式的可满足式。公式__E__是析取范式。
供选择的答案
A： ① 3 ② 8 ③ 27 ④ 64
B： ① 4 ② 16 ③ 64 ④ 256
C、D： ① ┐（P→q）∧q∧r
② p→(p∨q)
③ (p∧q) ∨(p∧┐q)
④ ┐(p∧q)→┐p∨┐q
E: ① ┐P∨(q∧-r) ∨(p∧r)q┐
② (p∨┐q)∧(q∨r) ∧┐r
③ (p∨q) ∧┐(q∨┐r)
④ (p→q) ∨(p∧r)

试题 15
从供选择的答案中，选出应填入下面叙述中__?__内的最确切的解答，把相应编号写在答卷的对应栏内。
n 个未知数的线性方程组的矩形式是
A x=b
其中 A 为 n 阶矩阵，b 和 x 为 n 维列向量。高斯一约当法是一种__A__。它是把矩阵 A 变换成__B__的解法，计算量
（乘法或除法）约为__C__。若在计算过程中某对线（例如第 K 行，第 K 列）上的元素是 0 或者接近于 0，就无法
求解或解的精确度很差，因此往往进行行或列交换，把比 X 大的行或列中__D__的元素交换到对角线上求解。即
使进行这种交换，对角线元素仍近似为 0 的矩阵称为 __E__。
供选择的答案
A： ① 直接法 ② 间接法 ③ 迭代法 ④ 牛顿法
B： ① 上三角矩阵 ② 下三角矩阵 ③ 单位矩阵 ④ 三对角线矩阵
C： ① ② ③ ④
D： ① 绝对值最大 ② 绝对值最小 ③ 最大 ④ 最小
E： ① 正定矩阵 ② 病态矩阵 ③ 奇异矩阵 ④ 希尔伯特矩阵

（1997 年度——下午试题）
试题一
阅读以下说明和流程图，回答问题 1 至问题 4，将解答写在答卷的对应栏内。
[说明]
某公司将其生产的商品通过若干个销售点进行销售。销售点在收到商品后的规定时间内把货款汇给公司。
流程图描述了该公司发货、收款、催款的处理过程。其中部分文件和单据的格式如下：
商品文件：商品代号，商品名称，单价
销售点文件：销售点代号，销售点名称，地址
发货单：发货日期，销售点代号，商品代号，数量，金额
收款单：收款日期，销售点代号，商品代号，数量，金额，该商品的发货日期
处理 1～3 把当天的发货单合并到发货文件。处理 4～6 把当天的收款单合并到收款文件。处理 7 在发货文件中当天已收款的记录上加上已收款标记。处理 8 和处理 9 在月末执行一次，主要用于输出月发货报、催款通知单、月收款报告。
[问题 1]
详细写出流程图中商品文件和销售点文件对处理 1 的作用。
[问题 2]
说明处理 8 为何要写发货文件。
[问题 3]
说明处理 9 除制作月收款报告外还对收款文件做什么操作。
[问题 4]
为了提高处理效率，流程图需作何改动。

试题二
阅读以下说明和流程图，回答问题 1 至问题 3，将解答写在答卷的对应栏内。
[说明]
本流程图用来实现一组正整数的加权舍位平衡。已知正整数组
A(a1,a2,…，an)

满足条件 a1= (n>3)。现将数组 A 中的每个数舍 P 位（P 为正整数）后，得到另一正整数数组
B(b1,b2,…,bn)
它满足如下条件：
1.b1 是 a1 舍 P 位后四舍五入所得，即 b1=INT(a1/10p0.5)
2.b1 =
3.bi=INT(ai/10P)+ei(i=2,3,…,n)，其中 ei 取值为 0 或 1，当 ei=1 时，称 ei 是第 i 项数据的进位
4.ei(i=2,3,……,n)之值根据余数 MOD（ai,10P）乖上权 fi(fi>0)后的数值大小来决定（其算法见流程图），
权 fi 存放在数组 F 中。
其中 INT 是取整数函数，MOD 是余数函数。例如正整数 78965 舍 P=3 位，则
INT（78965/103）=78
MOD（78965，103）=965
[问题 1]
填充流程图中①～④ ，把解答写在答卷纸的相应位置上。< br>[问题 2]
若 N=5，P=1，A=（1586，985，26，247，328）
F=（1，1，1，1，1）
则数组 B 的值是多少？
[问题 3]
若 N=3，P=1，A=（41，16，25），F=（1，0，0），则数组 B 的值是多少？

试题三
阅读以下说明和流程图,回答问题 1 至问题 3,将解答写在答卷的对应栏内。
[说明]
下面给出的是某房产管理系统的一套分层数据流图。其功能描述如下：
系统随时根据住房送来的入信单更新信户基本信息文件；
每月初系统根据物业管理委员会提供的月附加费（例如清洁费、保安费、大楼管理费等&#0;）表和房租调整表，计算每家住户的月租费（包括月附加费），向住户发出交费通知单。住户交费时，系统输入交费凭证，核对后输出收据给住户；
系统定期向物业管理委员会提供住房分配表和交费情况表；
住户因分户或换房，在更新住户基本信息文件的同时，系统应立即对这些住户做月租费计算，以了结分户或换房前的房租。
假定题中提供的顶层图是正确的，请回答下列问题：
[问题 1]
指出哪张图中的哪些文件可不必画出。
[问题 2]
指出在哪些图中遗漏了哪些数据流。回答时请用如下形式之一：

××图中遗漏了××加工（或文件）流向××加工（或文件）的××数据流；
××图中加工××遗漏了输入（或输出）数据流××。
[问题 3]
指出加工 2 图中加工 2.3 能检查出哪些不合格交费凭证。
[流程图]
顶层图

试题四
在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读下列程序说明和 CASL 程序，将应填入__(n)__
处的字句，写在答卷的对应栏内。
[程序说明]
本子程序将一个非负二进整数翻译成五位十进整数字符。
进入子程序时，在 GR0 中给出被翻译的非负二进整数，在 GR2 中给出存放五位十进整数数字字符的起始地址。
十进制数字字符用 ASCII 码表示。当结果小于五位时，左边无空白符替换；当二进整数为零时，在（GR2）+4 中存放 0 的 ASCII 码。
数字字符 0 至 9 的 ASCII 码是 48 至 57，空白符的 ASCII 码是 32。
[程序]

```
START
LEA GR1，0
LEA GR3，32
L1 ____(1)____
JPZ L2
ST GR3,0,GR2
LEA GR2,1,GR2
LEA GR1,1,GR1
JNZ L1
L2 ___(2)___
L3 ___(3)___
JMI L4
SUB GR0,SN0,GR1
LEA GR3,1,GR3
___(4)___
L4 ST GR3,0,GR2
LEA GR2,1,GR2
LEA GR1,1,GR1
___(5)___
JNZ L2
RET
SON NC 10000
DC 1000
DC 100
DC 10
```

```
DC 1
END
```

试题五

阅读以下程序说明和 FORTRAN 程序，将应填入__(n)__处的字句，写在答卷的对应栏内。

[程序说明]

对称矩阵通常只需存储其下三角部分，例如，下列对称矩阵可用一维数组（1，2，3，4，5，6，7，8，9，10）存储其下三角部分。N 阶对称矩阵下三角部分的元素个数为(N*N+N)/2。

本子程序用来计算 N 阶对称矩阵 A 的平方 B，B 也是一个 N 阶对称矩阵。程序中 X，Y 是分别存入 A，B 下三角部分的一维数组。

[程序]

```
SUBROUTINE P（X，Y，N）
INTEGER X（N*N+N）/2，Y（N*N+N）/2），S
M=1
DO 10 JJ=__(1)__
DO 10 II =__(2)__
I=II
J=JJ
L=__(3)__
S=0
DO 30 K=1,N
S=S+X(I)*X(J)
IF(____(4)____)THEN
I=I+L
ELSE
I=I+1
ENDIF
IF(____(5)____)THEN
J=J+L
ELSE
J=J+1
ENDIF
L=L-1
30 CONTINUE
Y(M)-S
M=M+1
CONTINUE
END
```

试题六

阅读以下程序说明和 C 程序，将应填入__(n)__ 处的字句，写在答卷的对应栏内。

[程序说明]

某系统由 n 个部件组成，这些部件被物理地分成若干个分离的部件组。同一组内的两件部件 i 和 j，它们或直接相连，或间接相连（部件 i 和部件 j 间接相连是指在这两件部件之间有一个部件相连序列，其中部件 i 和 j 分别与这相连序列中的某个部件直接相连）。系统的 n 个部件被统一编号为 0，1，…，n-1。本程序输入所有直接相连的部件号对，分别求出系统各分离部件组中的部件号并输出。程序根据输入的直接相连的两件部件号，建立 n 个链表，其中第 i 个链表的首指针为 s[i]，其结点是与部件 i 直接相连的所有部件号。程序依次处理各链表。在处理 s[i]链表中，用 top 工作链表重新构造 s[i]链表，使 s[i]链表对应系统中的一个部件组，其中结点按部件号从小到大连结。

[程序]

```
# include
#define N 100
typeef struct node{
int data;
struct node *link;
} NODE;
NODE*s[N];
int i,j,n,t;
NODE *q,*p,*x,*y,*top;
main()
{ printf(“Enter number of parts.”);
scanf(“%d”,&n);
for(i=0;idata=j; p->link=s[i]; s[i]=p;
p=(NODE *)malloc(sizeof(NODE));
P->data=i; p->link=s[j]; s[j];=p;
}
for(i=0;idata]!=NULL)
{ /将 j 链表也移入工作链表*/
for(p= s[j];p->link! =NULL;p=p->link);
p->link=top; top=s[j];____(3)____;< br>}<br>/*在重新生成的第 i 链表中寻找当前结点的插入点*/<br>for(y=s[i];____(4)____;x=y,y=y->link);
if(y!=NULL && y ->data= =q->data)
free(q);/*因重新生成的第 i 链表已有当前结点，当前结点删除* /
else { /* 当前结点插入重新生成的 i 链表*/
____(5)____;
if(y= =s[i]) s[i]=q;
else x->link=q;
}
}
for(i=0;idata);
q=p->link; free(p); p=q;
}
printf(“\n”);
```

```
}
}
```

试题七

阅读以下程序说明和 FORTRAN 程序，将应填入__(n)__处的字句，写在答卷的对应栏内。

[程序说明]

本子程序实现地图的着钯。在地图上，一个国家所着的颜色必须与所有相邻的国家所着的颜色不同。现已证明，仅需四种不同的颜色就能解决地图的着钯若地图上有 N 个国家，分别用 1 至 N 编号。子程序中用数组 INDEX（N，2）和 BORDER（M）存放 N 个国家的相邻情况。

INDEX（I，1）和 INDEX（I，2）分别表示与第 I 国相邻的国家编号在数组 BORDER 中的起始位置和终止位置，即这些邻国的编号存放在 BORDER（INDEX（I，1））至 BORDER（INDEX（I，2））中。例如，对应于图 1 所示的地图，数级 INDEX 和 BORDER 具有如下值：

INDEX=BORDER-（2，3，4，5，1，3，1，2，4，5，1，3，5，1，3，4）

子程序中分别用 1，2，3，4 代表四种不同颜色，着色结果存放在数组 COLOR 中，即数组元素 COLOR（I）的值为第 I 个国家所着的颜色。子程序采用试控法找解。首先从第 I 个国家着第一种颜色开始，顺序为各个国家寻找着色方案。对第 I 个国家，当为它找到一种未被它的相邻国家着色的颜色时，就用该颜色对此国家着色，并准备处理下一国家；当不能为它找到一个未被它的相邻国家着色的颜色时，就回溯&#0;&#0;即改变第 I-1 个国家的着色方案。直至最终为全部国家找到着色方案。

[程序]

```
SUBROUTINE P（INDEX，BORDER，COLOR，N，M）
INTEGER INDEX（N，2），BORDER（M），COLOR（N）
DO 10 I=1，N
COLOR（I）=0
I=1
IF（__(1)__）THEN
K=COLOR(I)+1
IF(__(2)__)THEN
J=INDEX(I,1)
IF(J.LE.INDEX(I,2))THEN
IF(__(3)__)THEN
J=J+1
GOTO 20
ELSE
K=K+1
____(4)____
ENDIF
___-(5)____
I=I+1
GOTO 40
```

```
ENDIF
COLOR9I)=0
_____(6)_____
GOTO 40
ENDIF
IF(I.EQ.0)THEN
WRITE(*,*)’ NO SOLUTION’
ENDIF
END
```

试题八

阅读以下程序说明和 C 程序，将应填入__(n)__处的字句，写在答卷的对应栏内。

[程序说明]

一个相连的区域被不规则地分割成 n 个不同的小区域；每个小区域与若干其它小区域相邻接。现用 cn 种不同颜色为该区域着色，要求每个小区域着同一种颜色，相邻小区域着不同颜色。

设小区域被顺序编号为 0，1，…，n-1。每个小区域与其它小区域的邻接关系用两维数组 bordering 表示，元素 bordering[i][j]表示 i 号小区域与 j 号小区域之间的邻接关系：

0 j 小区域与 i 小区域不邻接 bordering[i][j]= 1 j 小区域与 i 小区域相邻接程序中，把计算结果存入于两维数组 colored 中，颜色编号为 0,1,…,cn-1,元素 colored[coler][j]的含义是

0 j 小区域不用颜色 color 着色 colored[color][j]=1 j 小区域用颜色 color 着色

函数 colorcountry(bordering,colored,n,cn)根据所给的小区域邻接关系数组 bordering、小区域个数 n 、颜色数 cn，将找到的着色方案记录在数组 colored 中。函数采用试探法找解。首先从第一个小区域着第一种颜色开始顺序为各小区域找着色方案。对某个小区域，当为它找到一种未被它的相邻小区域着色的颜色时，就用该颜色对该小区域着色，并准备处理下一个小区域。当不能为某个小区域找到一个未被它的相邻小区域着色的颜色时，就回溯。如最终为全部小区域找到着色方案，函数返回 1；否则，函数返回 0。

程序假定小区域个数不超过 20，颜色数为 4。

[程序]

```
#include
#define n 20
#define CN 4
int colorcountry(int bordering[][N], int colored[][N], int n,int cn)
{ int color,used,i,c;
for(color=0;colorn;i++)colored[color][i]=0;
c =0; /*从第 1 个小区域开始*/
color =0; /*从着第 1 种颜色开始试控*/
while(cc;i++)
if(____(2)____)used=1;
if(!used)break; /*当前颜色未被相邻小区域着色*/
color++
}
if(!used)
{ /*找到一种可用颜色，用此色着色，并准备处理下一个小区域*/
```

```
____(3)____=1; color=0;
}else{/*未找到一种可用颜色，回溯*/
c--;if (c<0)return 0; /*发现没有解的情况*/ for(color=0;____(4)____;color++); ____(5)____=0 } }
return 1; } print(int colored[ ][n],int n,int cn) /* 输 出 结 果 */ { char
*colort[ ]={“RED”,”BLUE”,”GREEN”,”YELLOW”}; int color,i; for(color=0;colorn;i++)
if(colored[color][i]) printf(“\t%d”,i);
printf(“\n”);
}
}
int colored[CN][N],bordering[N][N];
main()
{ int c,i,j,n;
printf(“Enter number of areas.”); scanf(“%d”,&n);
printf(“Enter bordering:\n”);
for(i=0;in;j++)bordering[i][j]=0;
for(i=0;i0 to next).\n”,i);
scanf(“&d”,&j);
while(j>=0)
{ if(i != j)bordering[i][j]=bordering[j][i]=1;
scanf(“%d”,&j);
}
}
if(colorcountry(bordering,colored,n,CN))
print(colored,n,CN);
else printf(“No Solution.\n”);
}
```

一九九七年度高级程序员级上午试题答案

试题号	解 答					得 分
	A	B	C	D	E	
例 题				③		
试题 1	10	5	7	8	9	
试题 2	5	7	7	4	4	
试题 3	3	2	2	4	3	
试题 4	4	1	2	6	5	
试题 5	3*	5*	6*	7*	9*	
试题 6	2	5	1	1	4	
试题 7	4	4	6	2	5	
试题 8	3	6	4	7	5	
试题 9	3		6	3	1	
试题 10	8	6	4	2	1	

整理人：刘海云
整理时间：2003.05

试题 11	2	6	10	9	5	
试题 12	1	3	3	2	1	
试题 13	4	6	5	5	3	
试题 14	2	4	1	3	1	
试题 15	1	3	3	1	2	
评阅人		校阅人		小 计		

注：*表示可交换

一九九七年度高级程序员级下午试题答案

流程图

试题一（15 分）

（1）3 分 检查发货单中非法销售代号、非法商品代号、数量*单价≠金额等错误。

答“检查发货单的合法性”得 2 分

（2）4 分 “删除发货文件中已做收款标记的所有记录”或“将未收到款的记录重新组成新发货文件”。

（3）4 分 “删除收款文件中的所有记录”或“删除已收到款的记录”或“对收款文件初始化”。

（4）4 分 从收款文件到处理 7 的连线改成从日收款分类文件到处理 7 的连线。答“从收款文件到处理 7 的连线改成从日收款文件到处理 7 的连线”得 3 分，其它解答不给分。

试题二（15 分）

1. 9 分

① 2 分 i ：n

② 2 分 i ：n

③ 2 分 i ：n

④ 3 分 k ：G 或 K ： b1-E 答 K ： G-1 给 2 分。

（2）4 分 159，98，3，24，33

注：错 1 个扣 1 分，错 4、5 个不给分。

整理人：刘海云
整理时间：2003.05

（3）2 分 4，1，3 注：错 1 个不给分。

试题三（15 分）

1. （3 分） 0 层图中的房租文件和交费文件是局部文件，可不必画出。

   注：多写一个文件扣 1 分，少 1 个文件扣 2 分。

2. （8 分）

① 加工 1 子图中，遗漏了从住户基本信息文件到加工 1.1（入住单校验）的数据流。

② 加工 1 子图中，加工 1.6（制作住房分配报告）遗漏了输出数据流：住房分配表。

③ 加工 2 子图中，加工 2.1（计算月租费）遗漏了输入数据流：月附加费表。

3.加工 2 子图中，加工 2.4（制作收据）遗漏了输出数据流：收据。

注：答对 1 点得 3 分，答对 2 点得 6 分，答对 3 或 4 点得 8 分。未答下划线部分的内容时，另共扣 1 分。

1. （4 分）

① 2 分 交费凭证中有非法字符

② 2 分 交费文件中不存在与之对应的交费凭证

C

试题六（15 分）

1. （3 分）s [i] = NULL 答 s [i] =…给 1 分。
2. （3 分）top = top ->link 答 top =…给 1 分。
3. （3 分）s [j] = NULL 答 s [j] =…给 1 分。
4. （3 分）y ! = NULL && y -> data < q -> data 或 y && y -> data < q -> data

   答 y -> data < q -> data && y 或 y -> data < q -> data 给 2 分。

   答 y ! = NULL && y -> data ! = q -> data 给 2 分。

   答 y -> data != q -> data 给 1 分。

5. （3 分）q -> link = y

答 q -> link =x -> link 给 2 分。

答 q -> link =…给 1 分。

试题八（15 分）

1. （3 分） color < cn

   答 color < 4 给 3 分；答 color <= cn 给 2 分。

2. （3 分）bordering [c] [i] && colored [color] [i]

   答 bordering [c] [i] ==1 && colored [color] [i] == 1 给 3 分。

   答 bordering [c] [i] * colored [color] [i] == 1 给 3 分，而将其中相等运算符“==”写成赋值运算符“=”时，只给 1 分。其中 bordering [c][i]可写成 bordering[i][c]。运算符‘&&’左右只对一半给 2 分。

3. （3 分）colored [color] [c++]

   答 colored [color] [c]给 2 分。

   答 colored [color] […]给 1 分。

   答 c++给 1 分。

4. （3 分） colored[color][c] == 0 或! colored [color] [c]或 colored [color] [c] !==1
5. （3 分）colored [color ++] [c]

答 colored [color] [c]给 2 分，

答 colored [color] […]给 1 分。

CASL 汇编语言

试题四（15 分）

1. （3 分） COL GRO， SNO， GR1（用 CPA 指令也可）
2. （3 分） LEA GR3， 48

   答 LEA GR3， 0 给 1 分。

3. （3 分） CPL GRO， SNO，GR1（用 CPA 指令也可）
4. （3 分） JMP L3

   答 JPZ L3 或 JNZ L3 也可给 3 分。

5. （3 分） LEA GR4，-5，GR1

答 LEA GR4， -4， GR1 给 1 分

其中 GR4 可写成 GR3。

FOXBASE

试题六（15 分）

1. （3 分） 1，N
2. （3 分） JJ，N
3. （3 分） N-1
4. （3 分） K . LT . II
5. （3 分） K . LT . JJ

试题八

1. （3 分） I . GE . 1 . AND . I . LE . N

   答 I ． LE ． N 给 2 分，答 I ． GE ． 1 给 1 分。

2. （2 分） K . LE . 4 或 K . LT . 5
3. （3 分） K . NE . COLOR(BORDER(J))

   答 K ． NE ．…给 1 分，答…. NE ． COLOR(BORDER(J))给 2 分

4. （2 分） GOTO 30
5. （3 分） COLOR（I） = K
6. （2 分） I = I - 1

高级程序员计算机软件专业技术资格和水平考试

1998 年试题与答案

（1998 年度——上午试题）

试题 1

从供选择的答案中，选出应填入下面叙述中_？ _内的最确切的答案，把相应编号写在答卷的对应栏内。

设计高质量的软件是软件设计追求的重要目标。可移植性、可维护性、可靠性、效率、可理解性和可使用性等都是评价软件质量的重要方面。

可移植性是反映出把一个原先在某种硬件或软件环境下正常运行的软件移植到另一个硬件或软件环境下，使该软件也能正确地运行的难易程序。为了提高软件的可移植性，应注意提高软件的_A_。

可维护性通常包括_B_。通常认为，软件维护工作包括改正性维护、_C_维护和_D_维护。其中_C_维护则是了为扩充软件的功能或提高原有软件的性能而进行的维护活动。

_E_是指当系统万一遇到未预料的情况时，能够按照预定的方式作合适的处理。

供选择的答案

A：①使用方便性 ②简洁性 ③可靠性 ④设备不依赖性

B：①可用性和可理解性 ②可修改性、数据独立性和数据一致性
③可测试性和稳定性 ④可理解性、可修改性和可测试性

C、D：①功能性 ②扩展性 ③合理性 ④完善性
⑤合法性 ⑥适应性

E：①可用性 ②正确性 ③稳定性 ④健壮性

试题 2

从供选择的答案中，选出应填入下面叙述中_？_内的最确切的答案，把相应编号写在答卷的对应栏内。

实体联系模型(简称 ER 模型)中的基本语义单位是实体和联系。ER 模型的图形表示称为 ER 图。联系可以同_A_实体有关。实体与实体之间的联系可以是_B_。

利用 ER 模型进行数据库的概念设计，可以分成三步：首先设计局部 ER 模型，然后把各个局部 ER 模型综合成一个全局的模型，最后对全局 ER 模型进行_C_，得到最终的 ER 模型。

ER 模型向关系模型的转换南昌 把一个实体类型转换成一个关系模式,实体的属性是关系的属性，实体的键是关系的键；是一个联系类型转换成一个_E_，参与该联系类型的各实体的键以及联系的属性转换成_E_，其中的键由实体与实体之间的联系决定。

供选择的答案

A：①0 个 ②1 个或多个 ③1 个 ④多个

B：①一对一和一对多 ②一对一和多对多
③一对多和多对多 ④一对一、一对多和多对多

C：①简化 ②结构化 ③最小化 ④优化

D：①联系模式 ②数据模式 ③关系模式 ④逻辑模式

E：①联系属性 ②关系的属性 ③数据属性 ④关系的候选键

试题 3

从供选择的答案中，选出应填人下面叙述中_ ？ _内的最确切的解答，把相应编号写在答卷的对应栏内。

在内部排序中，通常要对被排序数据序列进行多趟扫描。各种排序方法有其不同排序实施过程和(时间)复杂性。

对给定的整数序列(541，132，984，746，518，181，946，314，205，827)进行从小到大的排序时，采用冒泡排序和直接选择排序时若先选出大元素，则第一趟扫描结果分别是_A_和_B_;采用快速排序(以中间元素 518 为基准)的第一趟扫描结果是_C_。

设被排序数据序列有 n 个元素，冒泡排序和直接选择排序的复杂性是一 D 一；快速排序的复杂性是_E_。

供选择的答案

A、 B、 C：① (181，132，314，205，541，518，946，827，746，984)
② (541，132，827，746，518，181，946，314，205，984)
③ (205，132，314，181，518，746，946，984，541，827)
④ (541，132，984，746，827，181，946，314，205，518)
⑤ (132，541，746，518，181，946，314，205，827，984)
⑥ (132，541，746，984，181，518，314，946，205，827)

D、 E：① O(n log2n) ② O(n) ③ O(log2n) ④ O(n2)
⑤ O((10g2n)2) ⑥ O(n2logln)

试题 4

从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

语法分析方法大体上可分成自上而下和自下而上两种。自下而上分析法，是从输符号串开始逐步进行_A_，直至_A_成文选的起始符号。自上而下分析接，则是从文选法的起始符号开始反复使用产生式进行_B_直至_B_出输入符号串。

算符优先文法是一种自下而上分析方法，其文法的特点是文法的产生式中一C 一。自上而下的分析方法，通常要求文选的产生式_D_ ，如_E_文法就是一种可以自上而下分析的文法。

供选择的答案

A、B：①递归 ②综合 ③回归 ④推导
⑤分解 ⑥归约

C：①不含两个相邻的非终结符 ②不含两个相邻的终结符
③不含 ε 产生式 ④不含长度为 1 的产生式

D：①不以非终结符开头 ②不以终结符开头
③不含左递归 ④不含右递归

E：① LR(I) ② LL(I) ③ SLR(I) ④LALR(I)

试题 5

从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

信号量是操作系统中用作互斥和同步机制的一个共享的整数变量。信号量仅可以由初始化、唤醒(Signal)和等待(Wait)三种操作访问。

对于给定的信号量 S，等待操作 Wait(S)(又称 P 操作)定义为

if S＞0 then_A_ else 挂起调用的进程

唤醒操作 Signal(S)(又称 V 操作)定义为

if 存在等待的进程 then 唤醒这个进程 else _B_

给定信号量 S，可以定义一个临界区来确保其互斥、即保证在同一时刻这个临界区只能够被一个进程执行。当 S 被初始化为 1 时，代码段

_C_ ；
{临界区}
_D_

定义了一个临界区。

这样的临界区实际上是将共享数据和对这些数据的操作—起封装起来、通过

其互斥机制一次只允许一个进程进入，这种临界区通常称为_E_。
供选择的答案
A— D：①S：＝0 ②S：＝S＋1 ③S：＝S—1 ④S：＝1
⑤Signal(S＋1) ⑥Wait(S－1) ⑦Signal(S) ⑧Wait(S)
E：①模块 ②类程 ③管程 ④线程

试题 6
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
在高级程序设计语言中，使用参数是子程序之间传递信息的一种手段。子程序说明中的参数称为形式参数，调用语句中的参数称为实在参数。调用时，实在参数的个数、类型和顺序要和形式参数保持一致。
知道一种语言(或编译器)使用哪种参数传递方法是很重要的，因为程序的运行依赖于所用的方法。参数传递方法有传值调用(call by value)、引用调用(call by reference)、传名调用
(Call by name)和宏扩展(macro expansion)。
传值调用是指把实在参数的_A_传递给相应的形式参数、子程序通过这种传值形参_B_；引用调用是指把实在参数的_C_传递给相应的形式参数，此时子程序形式参数的一次引用或赋值被处理成对形式参数的_D_访问。
C 语言中的函数，以_E_方式进行参数传递。
供选择的答案
A、C：①地址 ②名 ③值 ④地址和值：
⑤值和名 ⑥名和地址
B：①可传回结果的值 ②可传回存放结果的地址
③可传回结果的值和存放结果的地址 ④不可传回任何结果(值或地址)
D：①直接 ②间接 ③变址 ④引用或赋顿
E：①传值调用 ②引用调用 ③传名凋用 ④宏扩展 试题 7
从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
虚拟存贮管理系统的基础是程序的_A_理论。这个理论的基本含义是指程序执时往往会_B_访问内存贮。程序的_A_表现在_C_和_D_上。_C_是指最近被访问的存贮单元可能马上又要被问。_D_是指马上被访问的单元，而其附近的单元也可能马上被访问。
根据这个理论，Denning 提出了工作集理论。工作集是进程运行时被频繁地访问的页面集合。在进程运行时，如果它的工作页面都在_E_器内，能够使该进程有效地运行，否则会出现频繁的页面调入／调出现象。
供选择的答案
A：①局部性 ②全局性 ③动态性 ④虚拟性
B：①频繁地 ②均匀地 ③不均匀地 ④全面地
C、D：①数据局部性 ②空间局部性 ③时间局部性 ④数据全局性
⑤空间全局性 ⑥时间全局性

E：①外部存贮 ②主存贮 ③辅助存贮 ④虚拟存贮

试题 8

从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

设有三个指令系统相同的处理机 X、Y 和 Z，它们都有 4K 字节的高速缓冲存贮器(Cache)和 32M 字节的内存，但是其存取周期都不一样，如下表所示(Tic 和 Tim 分别表示 i 处理机 Cache 存取周期和主存存取周期)；

X

Y

Z

Tic

40ns

100ns

120ns

Tim

1μs

0.9μs

0.8μs

若某段程序，所需指令或数据在 Cache 中取到的概率为 P＝0．5，则处理机 X 的存器平均存取周期为 _A_μS。并假定指令执行时间与存贮器的平均存取周期成正比此时三个处理机执行该段程序由快到慢的顺序为_B_。

若 P＝0．65 时，则顺序为_C_。

若 P＝0．8 时，则顺序为_D_

若 P：0．85 时，则顺序为_E_。

供选择的答案

A：①0．2 ②0．48 ③0．52 ④0．6

B～E：①X、Y、Z ②X、Z、Y ③Y、X、Z ④Y、Z、X

⑤Z、X、Y ⑥ Z、 Y、 X

试题 9

写在从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

在段页式管理的存贮器中，实存等分为_A_、程序按逻辑模块分成_B_。D 在多道程序

环境下，每道程序还南要一个_C_作为用户标志号。每道程序都有对应的_D_。一个逻辑地址包括_C_x。段号 s、页号 p 和页内地址 d 四个部分。

假设总长度为 22 位的逻辑地址格式分配如下：21—20 位 X；19—14 位 s；13—11 倍；若 p；10－0 倍 d。若 x， s， p， d 均以二进制数表示，其转换成的物理地址为_E_。

供选择的答案

A—C：①段 ②页 ③基 ④模块
⑤区域 ⑥段号 ⑦页号 ⑧基号
⑨模块号 ⑩区域号

D：①一个段表和一个页表 ②一个段表和一组页表
③一组段表和一个页表 ④一组段表和一组页表

E：① x×220 十 s×214 十 p×211 十 d ② ((x)十 x 十 p)×211 十 d
③ (((x)十 s)十 p)十 d ④ (((x)十 s)十 p)×212 十 d

[注]式中(Y)表示地址为 Y 的单元的内容

试题 10

从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

有多种设备可以实现不同网段或网络之间的互连，互连设备通常可按工作在 OSI 模型中的层次来划分。在物理层实现互连的称为 _A_；在数据链路层实现互连的称为_B_；在网络层实现互连的称为_C_；在运输层及以上高层实现互连的设备称为网关或_D_。_E_也是一种用来构造局域网的常用设备，通常可以用双绞线把服务器与 PC 客户机等连入_E_。

供选择的答案

A—E：①集线器 ②协议转换器 ③网桥 ④路由器
⑤网关 ⑥转发器试题 11

写在从供选择的答案中，选出分别与下列概念最密切相关的术语组，把相应编号写在答卷的对应栏内。

1．电子商务 _A_ 2．人机界面 _B_
3．计算机性能评价 _C_ 4．并行处理_D_
5．网络体系结构 _E_

供选择的答案

A—E：①OSI、对等层协议、无连接服务 ②OSI、对等层协议、TPC
③OCR、基准测试程序、 TPC ④吉普森方接、基准测试程序、TPC
⑤EDI、网上商店、身份验证 ⑥向量计算机、MPP、多指令流
⑦ EDI、网上商店、OCR ⑧所见即所得、语音识别、OCR
⑨ EDI、MPP、多指令流 ⑩语音识别、OCR、OSI

试题 12

从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

Many word processing programs include spell checker． It checks the spelling of every word in a _A_ by looking up each word in its dictionary． If the word does not appear in the dictionary the user is _B_ to a possible misspelling and possiblle corrections are often _C_ Spell checker dies not recognize unusual people names or specialized terms,but it wi11 often allow you to create your own personal dictionary of specialized words you often use．Spell checker is avaluable aids

to proofreading， but it can not catch the_D_ of one correctly spe11ed word for another (such as form for from) . Thus it does not _E_ a document is free of spelling errors.

供选择的答案

A：①document ②equipment ③program ④statement

B，C：①alerted ②alternate ③guessed ④guided

⑤Suggested ⑥surprised

D：①addition ②condition ③notation ④substitution

E：①committee ②correct ③guarantee ④prove

试题 13

从供选择的答案中，选出应填人下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

A local-area network (LAN) is a communications network that _A_ a variety of device and provides a _B_ for information exchange among those devices. The scope of the LAN is small typically a single building or a cluster of buildings. The LAN is usually owned by the same organization that owns the _C_ devices. The internal data rate of LAN is at least several Mbps. The basic Of LAN communication is broadcasting. At each station, there is a transmitter / receiver that communicates over a _D_ shared. by other stations. A transmiss from any one station is _E_ to and received by all other stations.

供选择的答案

A、 B、 D、 E：①broadcast ②cable ③control ④interconnects

⑤internet ⑥meander ⑦means ⑧medium

⑨moderm ⑩output switch relay

C：①adopted ②attached ③selected ④unified

试题 14

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的答案，把相应编号写在答卷的对应栏内。

1．设 S＝{{1，2，3}，{4,5}，{6，7，8}、那么_A_式为真。

2．某校足球队有球衣 30 件，蓝球队有球衣 15 件，排球队有球衣 18 件，三队队员总数为 50 人，其中有 3 人同时参加 3 个队，那么同时只参加两个队的队员有_B_2 。

3．设 S＝{a，b，c}上的关系 R1，R2、R3、R4，那么_C_具有传递性。

4．如果二无关系 R 是自反的、对称的和传递的，则 R 是_D_。

5．设 A＝{a，b，c}，B＝{1，2}，作 f：A→B，则不同的函数个数有_E_个。

供选择的答案

A：①{1，2，3}S ②{{4，5}}S ③1∈S ④φ∈S

B：①7 ②10 ③13 ④16

C：①R1＝{〈a，c〉，〈c，a〉，〈a，b〉，〈b，a〉}

②R2＝{〈a，c〉，〈c，a〉，〈a，a〉}

③R3＝{〈a，c〉，〈c，c〉，〈b，a〉，〈b，c〉}

④R4＝{〈a，a〉}

D：①全序关系 ②等价关系 ③偏序关系 ④似序关系

E：①5 ②6 ③8 ④9

试题 15

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的答案，把相应编号写在答卷的对应栏内。

1．函数方程 g(x)=0 的解 x 称为方程的根，若 g(x)是次数≥ _A_的多项式，其求根问题一般不能用公式表示，只能用数值方法求解。

2．方程 x3-x2-1=0 有一实根，用二分法求根时，初始区间应取_B_。

3．用弦截法计算的迭代公式是_C_，用牛顿法计算的迭代公式是_D_。

4．对于单根，牛顿法至少_E_阶收敛。

A：①3 ②4 ③5 ④6

B：①[－1，0] ②[0，1] ③[1，2] ④[2，3]

C、D：①xn+1=2xn－ax2n ②xn+1=(xn+a/xn)/2

③xn+1=xn[(3/2)－(ax2n/2)] ④xn+1=(xnxn-1+a)/(xn+xn-1)

⑤xn+1=[(2xn)+(a/x2n)]/3 ⑥xn+1=(xnxn-1－a)/(xn－xn-1)

E：①1 ②2 ③3 ④4

（1998 年度——下午试题）

试题一

阅读以下说明和流程图，回答问题 1 至问题 3，将解答写在答卷的对应栏

[说明]

本流程图描述某超市销售数据的部分处理流程。超市中有若干台收款机和若干名收款员。这

里，我们把一个收款员开始使用一台收款机到离开这台收款机称为该收款员的一次作业。作业开始时，收款员先在收款机上输入收款员号和作业前金额。作业前金额是为了销售时的找零而在作业前预先放入钱箱的金额数。作业结束时，收款员要找开钱柜，取走全部现金，并把这些现金的金额数(称为作业后金额)输入收款机。当

作业前金额＋本次作业售货总金额－本次作业退货总金额≠作业后金额

时，表示这次作业存在金额差错。

本流程图已作简化，并作以下假定；该超市只有现金交易(不用信用卡和礼券)；一个收款员因某种原因(如吃饭)在一天中可以有多个作业；销售方式只有售货和退货两种。

整个超市分成若干部门(如食品部、服装部)，系统按部门统计一个月中各类货物的销售数量和金额，最后根据月销售计划文件分析各部门完成销售计划的情况。系统还统计每个收款员的差错情况和退货情况。

图中处理 4 和处理 8 每月的最后一天执行一次(营业结束后)，其它处理每天执行一次。

图中部分数据、文件的记录格式如下：

日销售数据：收款机号十收款员号十作业前金额十

{(售货标记|退货标记)十货号十数量十单价十金额}＋作业后金额

日销售文件记录：(作业开始标记十收款机号十收款员号十作业前金额)

|(售货标记|退货标记)十货号十数量十金额))

|(作业结束标记十收款机号十收款员号十作业后金额

部门日销售文件记录：部门号十(售货标记|退货标记)十货号十数量＋金额

部门月销售计划文件记录；部门号十月计划金额

收款员差错月报：月份十收款员号十差错作业数十差错总金额

收款员退货月报.. 月份十收款员号十退货次数十退货总金额

其中 {w}表示 w 重复出现多次； a|b 表示 a 或 b； a＋b 表示 a 与 b。

[问题 1]

分别写出收款员日销售文件、商品文件、部门日销售汇总文件至少应包含哪些数据项。

[问题 2]

处理 1 能检查出日销售数据中的哪些错误。

[问题 3]

处理 4 对收款员月销售文件作何种操作。

试题二

阅读以下说明和流程图，回答问题，将解答写在答卷的对应栏内。

[说明]

本流程图描述了某子程序的处理流程，现要求用白盒测试法为该子程序设计测度数据

[问题]

根据判定覆盖、条件覆盖、判定一条件覆盖、条件组合覆盖(即多重条件覆盖)、路径覆盖等五种覆盖标准,从供选择的答案中分别找出满足相应覆盖标准的最小测试数据组(用①～回答)。 供选择的答案

①x＝90，y＝90 x＝70，y＝70	②x＝90，y＝70 x＝70，y＝90	③x＝90，y＝90 x＝90，y＝75 x＝75，v＝90	④x＝90，y＝75 x＝75，y＝90 x＝70，y＝70
⑤x＝90，y＝90 x＝90，y＝75 x＝75，y＝90 x＝70，y＝70	⑥x＝80，y＝80 x＝90，y＝70 x＝70，y＝90 x＝70，y＝70	⑦x＝80，y＝80 x＝90，y＝75 x＝90，y＝90 x＝75，y＝90 x＝70，y＝70	⑧x＝80，y＝80 x＝90，y＝70 x＝70，y＝90 x＝70，y＝70 x＝70，y＝75
⑨x＝80，y＝80 x＝90，y＝75 x＝90，y＝70 x＝70，y＝80 x＝70，y＝75 x＝70，y＝70	⑩x＝90，y＝90 x＝90，y＝75 x＝90，y＝70 x＝75，y＝90 x＝70，y＝70 x＝70，y＝90	x＝80，y＝80 x＝90，y＝75 x＝90，y＝70 x＝70，y＝80 x＝70，y＝75 x＝70，y＝70 x＝70，y＝90	x＝80，y＝80 x＝80，y＝70 x＝70，y＝80 x＝70，y＝70 x＝90，y＝75 x＝90，y＝70 x＝70，y＝75 x＝75，y＝90 x＝75，y＝80 x＝70，y＝90

试题三

阅读以下说明和流程图，回答问题 1 至问题 2，将解答写在答卷的对应栏内。

[说明]

本流程图描述了某种字符序列的变换过程。变换前的字符序列以字符“＃”结尾，它存放在数组 old 中，变换后的字符序列存放在数组 new 中，它不包含结束标志“＃”。流程图按下列规则进行变换：

1．删除字符序列中的注解。注解用一对“!”字符分隔，注解中可出现除“＃”外的任何字符。

2．字符常数完整地保留。字符常数用一对“＆”字符分隔,字符常数中可出现、外的任何字符。值得注意的是：当字符“＆”需要出现在字符常数中时，通常用二个连续的“＆”表示，在保留字符常数时，这二个连续的“＆”都被保留。此外，作为字符常数分隔符的一对“＆”也被保留。

3．除字符常数和注解外，出现在其它地方的连续多个空格压缩成一个空格，但字符序列中先导

的空格则全部删除。

4．注解和字符常数之外的非空格字符均保持不变。

本流程图对原字符序列从左到右扫描，根据遇到的当前字符来搜索连续空格、注解或字符常

数，然后按上述变换规则变换。若遇到的当前字符是“！”，则寻找下一个“！”字符（若找不到则作出错处理），这二个“！”间的字符全部看作为注解。若当前字符是“&”字符则寻找与之配对的下一个“&”字符（若找不到则作出错处理），其间的字符全部看作为字

符常数。

例如：

原字符序列：

␣␣a␣␣␣b&cd&&␣␣！e&f␣␣g！h& ␣␣！␣␣j#

变换后的字符序列：a␣bdcd&&␣␣！e&f␣g␣j

本流程图假定在数组 old 中一定存在“#”字符。

[问题 1]

填充流程图中的①～⑤，把解答写在答卷的相应位置上。

[问题 2]

当原字符序列中汪解的前后均是连续空格时，本流程图将注解前后的连续空格分别压缩成一个空格，删除注解后，将导致变换后的新字符序列出现二个连续的空格。如；

g␣␣!h& ␣␣i!␣␣j

本流程图将变换成 g␣␣j。

为使变换后的新序列中除字符常数外没有连续的空格，图中的虚线框需作何改动(只需画出修改后的流程图)

试题四

在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读以下程序说明和 CASL 程序，将应填入_(n)_处的字句，写在答卷的对应栏内。

[程序说明]

本程序是统计字符串中数字字符“0”至“9”的出现次数。

字符串中的每个字符是用 ASCII 码存贮。一个存贮单元存放两个字符,每个字符占 8 位二进位。

程序中，被统计的字符串从左至右存放在 STR 开始的连续单元中，并假定其长度不超过 200，字符串以‘·’符作为结束。NCH 开始的 10 个单元存放统计结果。

```
      START MIN
MIN   LEA GR2，9
      LEA GR0，0
L1    _(1)_
      LEA GR2，－1，GR2
      LEA GR4，0
      LEA GR1，0
L2    LD  GR2，STR，GR1
      EOR GR4,C1
```

```
        JNZ RL
        _(2)_
RL      SRL GR2,8
        LEA GR3，0，GR2
        SUB BR3，C9
        JNZ L4
L3      LEA GR3，0，GR2
        SUB GR3，CO
        JM1 L5
        LEA GR2，1
        _(3)_
        _(4)_
L4      GR4 0，GR4
        JNZ L2
        _(5)_
        JMP L2
L5      SUB GR2，C
        JNZ L4
        EXIT
C1      DC 1
C       DC ’&middot;’
CO      DC ’0’
C9      DC 9
STR     DS 200
NCH     DS 10
        END
```

试题五

阅读以下程序说明和C程序，将应填入_(n)_处的字句，写在答卷的对应栏内。

[程序说明]

这里给出的程序逐一从指走课程成绩文件中读入学生的学号和成绩，对同一学生汇总他的总成绩，并按以下格式输出名次（按总成绩由高到低的顺序）、总成绩、同一名次的学生人数、同一名次学生的学号（按学号由小到大的顺序）。

程序约定学生学习课程不超过30种，课程成绩文件的第一个数字就是课程号，统计过程中，同一课程号的成绩文件不能重复输入。

程序采用链表结构存储学生有关信息，链表中的每个表元对应一位学生。程序在数据输入过程中，形成一个按学生学号从小到大顺序链接的有序链表。当数据输入结束后，程序按总成绩从高到低，学号从小到大的顺序对链表排序。程序最后按指定格式输出链表中的信息。程序的输出格式如下例所示：

名　次

总成绩

人　数

学　号

1

470

2

12

25

3

450

3

15

24

50

6

430

1

14

7

401

3

13

18

45

…

[程序]

```
#include 〈stdio.h〉
#define M 30
#define NLEN 10
typedef struct node {int cur_s / *最近输入成绩的科目 */
                     char no[NLEN]; int score;
                     stuction node *next;
                       } NODE;
NODE*bubblesort(NODE*head)
{NODE *q, *tail, *p＝(MODE*)malloc(sizeof(NODE);
  p→next＝head; head＝p; tail＝NULL;
  while (tail!=___(1)___)
  {p=head;q=p→next;
    while (q→enxt! =tail)
    {if (p→next→score→ 〈q→next→score||
         p→next→scor＝＝q→next→score＆＆
         strcmp(p→next→no, q→next→no)〉 0)
      {p→next＝_(2)_, /*两相邻表元链接关系前后颠倒*/
        _(3)_＝q→next→next; p→next→next＝q;
      }
      p＝p→next; /*调整 p 和 q*/ q＝_(4)_;
    }
    tail＝q;
  }
  p＝head→next; free(head); return p;
}
```

```
int s[M], sp;
main()
{FILE*fp;
  NODE*h, *u, *v*p;
  int ss, , mark, order, c;
  char fname[80], no[NLEN], ans;
  for(h=NULL, sp=0; ; )
  {print(''输入科目成绩文件名(输入 aaaa 表示强行结束)。\n");
    while (1)
    {scanf("%s",fname);
      if(strcmp(fname, "aaaa")==0)break;
      if((fp=fopen(fname, "r"))==NULL)
        printf("不能打开文件%s, 请重新输入科目文件名。\n, fname);
      else break;
    }
    if(strcmp(fname, "aaaa")==0)break;
    fscanf(fp, "%d", &ss); /&middot;输入科目号&middot;/s[sp]=ss;
    for(i=0; s[i]! =ss; i++);
    if(i<sp)
    {printf("该科目的成绩已输入, 请输入别的科目成绩文件。\n");
      continue;
    }
    sp++;
    while(fscanf(fp,"%s%d", no, &mark)==2)
    {/*在链表中寻找最近输入的学号*/
      for(v=h;v!=NULL,&& strcmp(v→no,no)<0;u=v,v=v→next);
      if(v! =NULL && strcmp(v→no, no)==0)
      {if (v→cur_s! ==ss)
        {v→score+=mark; v→cur_s=ss;
        }/*同一科目成绩的重复输入, 后输入成绩被忽略*/
      }else {p=(NODE*)malloc(sizeof(node); /*一位新的学生*/
              strcpy(p→no, no); p→score=mark; p→cur_s=ss;
              p→next=v;
              if(v==h) h=p; else u→next=p;
            }
    } fclose(fp);
    printf("还有科目成绩文件要输入吗? (Y/N)"); scanf("%c",&ans);
    if(anns=='N'||ans=='n')break;
  }
  h=bubblesort(h);
  printf("名次 总成绩 人数 学号\n"); /*以下按格式要求输出*/
  v=h; order=1;
  while(v! =NULL)
  {for (c=1,u=v→next;u!NULL &&u→score==v→score;c++,u=u→next);
```

```
    printf("%4d%7d%8d ", order, v→score, c);
    for(order+=c, i=1; _(5)_; v=v→next, i++)
    {if (i>1 && i%5==1) printf("\n%23c", '');
      printf("%s", v→no);
    }printf("\n");
  }
}
```

试题六

阅读以下程序说明和 FORTRAN 程序，将应填入_(n)_处的字句，写在答卷的对应栏内。

[程序说明]

本程序用以将 m+n 个元素的数组 A 中的前 m 个元素(A(1)-A(m))与后 n 个元素(A(m+l)-A(m+n))互换位置，并保持其各自原有的内部顺序。

程序中把数组的各元素看成首尾相连的序列,并将数组元素分成 G 组(G 为 m 和 n 的最大公因子)，每组中的元素序列是从数组某元素起由间隔为 m 的元素组成，再将每组元素依次循环向左传送，即得所需结果。

例如原数组 A 为：

由 m=4，n=2，求得 G=2，将 A 中的元数分成两组。按上述说明，第一组元素序列依次为 A(1)、A(5)、A(3)，将它们循环向左传送后的结果为

8

9

7

2

6

4

第二组元素序列依次为以 A(2)、A(6)、A(4)，将它们循环向左传送后的结果为

8

4

7

9

6

2

即得所需结果。

```
      INTEGET A(1000),  G, H
      READ(*, *)M, N
      READ(*, *)(A(1), 1＝1, M＋N)
      G＝M
      H＝N
20    IF (G, NE, H)THEN
        IF(G, GT, T)THEN
          G＝G－H
        ELSE
          H＝H－G
        ENDIF
        GOTO 20
      ENDIF
      DO 100 I＝_(1)_
        TEMP＝A(1)
        K＝1
30      J＝_(2)_
        IF(J, NE, 1)THEN
          A(K)＝_(3)_
          K＝J
          _(4)_
        ELSE
          A＝K＝_(5)_
        ENDIF
100   CONTINUE
      WRITE(*, 200)(A(I), I=1, M+N)
200   FORMANT(1016)
      END
```

试题七

阅读以下程序说明和 C 程序，将应填入_(n)_处的字句，写在答卷的对应栏内。

[程序说明]

本程序的函数

sum(int，i int total，int sigma，int rear，int d[]，int n)

用来从已知数组 d 的前 n 个元素中找出所有部分元素序列之和等于 total 的元素序列，约定数组 d 的元素都是正整数，且都小于等于 total。

函数 sum 使用递归方法找出全部解答。参数 i 表示递归函数当前考虑元素 d[i]，参数 sigma 是调用前已选取的部分序列的元素和，参数 rear 是后面还未考虑的那部分元素的元素和。

函数对元素 d[i]有两种可能的选择方案：

1．考虑元素 d[i]被包含在新的部分元素序列中的可能性。如果在当前部分元素序列之后接上 d[i]，新序列的元素和不超过 total，则函数将 d[i]包含在当前部分元素序列中。如果

新的部分元素序列的元素和等于 total 时，新的部分元素序列就是一个解答，函数将其输出；否则，若继续考虑后面的元素还有可能找到解答时，函数就递归去考虑后面的元素，寻找解答。最后，函烽就恢复原来部分元素序列中不包含 d[i]的状态。

2. 考虑元素 d[i]不被包含在新的部分元素序列中的可能性。如果继续向 d[i]之后考虑还是有希望能得到和为 total 的部分元素序列时，函数将新序列不包含 d[i]也作为一种可能的选择，并递归去考虑后面的元素，寻找解答。

```
#include 〈stdio.h〉
#define N 100
int a[N];
int flg[N];
sum (int i, int total, int sigma, int rear, int d[], int t)
{int j;
  /*考虑元素 d[i]被包含在新的部分元素序列中的可能性 */
  if(sigma+d[i] total/*如果 d[i]与当前序列的和不超过 total*/
   {flg[i]=1; /*d[i]被考虑在被部分元素序列中 */
    if(_(1)_==total)
    {/*输出解*/
      for (j=0; flg[j]==0; j++);
      printf("%4d=%d", total, d[j]);
      for(j++; <=i; j++)
          if (flg[j])
              printf("+%d",d[j]);
      printf("\");
    }
    else /*并且继续考虑后面的元素有可能找到解答时 */
    if(i<n-1 &&rear-d[i]+sigma>=total)
        sum(i+1, total, _(2)_, rear-d[i], d, n);
    _(3)_;
  /*考虑元素 d[i]不被包含在新的部分元素序列中的可能性。*/
  if (i<n-1&&rear-d[i]+tigma>=total)
          sum(i+1, total, _(4), rear-d[i], d, n);
}
main()
{int i, j, n, total, s, d;
  printf("输入 total! /n"); scanf("%d",&total);
  printf("输入 n! /n");  scanf("%d",&n);
  for (s=i=0; i<n; )
  {printf ("输入第%d 个元素>0 且<=%d)\n", i+1, total);
    scanf ("%d", &d);
    if( d<1||d>total)
      {printf("出错, 请重新输入! \n");
          contnue;
      }
```

```
        s＋＝a[i＋＋]＝d;
    }
    sum(0, total, 0,_(5)_, a, n);
    printf ("\n\n");
}
```

试题八

阅读以下程序说明和 FORTRAN 程序，将应填入_(n)_处的字句，写在答卷的对应栏内。

[程序说明]

本程序用以查找给定数组 A 的第 K 小元素。程序执行后，第 K 小元素存放在数组元素 A(K)中。

设给定的数组中各元素 A(i)＝(i＝1，2…N)互为不相等。它的第 K(1≤k≤N)小元素是指这样的元素；它大于 N 个元素中的 K－1 个元素，而小于其余的 N－K 个元素。假如，假设 N=8，数组 A 为：

44

94

12

50

55

6

18

67

它的第 5 小元素(即 K＝5)是 50。

本程序的处理过程如下：

将 A(K)的值(此例为 A(5)，此时值为 55)存入 X，从左向右扫描，直至遇以第一个不小于 X 的元素(此时为 94)，再从右向左扫描，直到遇见第一个不大于 X 的元素(此时为 18)，然后交换这两个元素，数且 A 成为：

44

18

12

50

55

6

94

67

继续这一“扫描与交换”的过程，直到两个方向的扫描在数组中间的某处汇合。此时数组被分成两段，汇合点左边的元素均小于汇合点右边的元素。然后选取存在第 K 小元素的一段，重复上述扫描、交换和调整扫描范围的处理过程。程序中采用 L、R 来表示扫描的范围的左右边界。开始时 L 为 1，R 为 N，以后逐次调整。当变成为 L 值不小于 R 值时，A[K]即是第 K 小元素。

```
      SUBROUTINT FIND(A, N, K)
      INTEGET A(N), R, X, W
      L＝1
      R＝N
20    IF (L, LT, R)THEN
        X＝A(K)
        I＝L
        J＝R
30      IF(A(1), LT, X)THEN
          I＝I＋1
          GOTO 30
        ENDIF
40      IF(___(1)___)THEN
         ___(2)___
          GOTO 40
        ENDIF
        IF(I,LE,J)THEN
          W＝A(I)
          ___(3)___
          I＝I＋1
          ___(4)___
        IF(I, LE, J)___(5)___
        ENDIF
        IF(J, LT, K)L＝I
        IF(___(6)___) R＝___(7)___
        GOTO 20
      ENDIF
      RETURN
      END
```

ASL 汇编语言文本

一、处理机 COMET 说明

1．COMET 是一台字长为 16 位的定点计计算机，主存储器的容量是 65536 字，按编号 0000～FFFF(十六进制)编址。

2．一个字的 16 位二进位采用自左至右的次序编号，即

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

3．一个字中的 16 位可以是：

不带符号的二进制非负整数，此时一个字能表示的数的范围是 0≤x≤2－1

用补码表示的带符号的二进制整数，此时一个字能表示的数的范围是：

地址常数，此时一个字能表示的地址写成十六进制时是：0000～FFFF

字符数据，此时一个字的高 8 位皆应为零，低 8 倍为字符的 ASCII 编码，即

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

0

0

0

0

0

0

0

0

字符的 ASCII 编码

4．COMET 具有 5 个通用寄存器 GR(16 位)，一个指令计数器 PC(16 位)和一个标志寄存器 RF(2 位)。它们的作用分别是：

GR(通用寄存器)。5 个通用寄存器的编号为 0，1，2，3，4 分别记为 GR0，GRL，GR2，GR3，GR4。这些通用寄存器用于 算术、逻辑，比较，移位等运算，其中 GR1，GR2，GR3、GR4 通用寄存器兼作变址寄存器。另外，GR4 还兼作栈指针(SP)用，栈指针是存放栈地十用的寄存器。

PC(指令计数器)，在执行指令的过程中，PC 中存放着正在执行的第一个字的地址(一条指令占二个字)。当指令执行结束时，置入下一条将要被执行的指令的第一个字的地址。也就是是说，在指令执行结束时，一般地把 PC 的内容加 2，只有在执行转移指令且转移条件成立时，才将转移地址置入 PC 中。

FR(标志寄存器)。在 ADD，SUB，AND，OR，EOR，CPA，CPL，SLA，SRA，SLL，SRL，LEAT 等指令执行结束时，根据执行结果，将 FR 置成 00、01 或 10 它不会因其它指令的执行而改变(参阅 7)。

5．COMET 的控制方式为顺序控制，指令由 32 位二进位构成，即双字长，其构成没有定义。

6．指令的符号表示：

GR 泛指通用寄存器，可用数字 0、1、2、3、4 来指定，也可以用 GR0、GR1、GR2、GR3、GR4 来指定。

XR 泛指变址寄存器，可用数字 1、2、3、4 来指定，也可以用 GR1、GR2、GR3、GR4 来指定。

SP 栈指针(第 4 号通用寄存器)。

ADR 标号(表示与标号对应的地址)，为 10 进制整数(－32768 ADR 65535；ADR 作为地址；具有 0～65535 的值，而 32768～65535 的值也可以用负的 10 进制常数表示)。

(X)个 X 地址中的内容，如果 X 是通用寄存器时表示通用寄存器中的内容。

E 表示有效地址，形成有效地址 E 的规则是：E＝ADR[＋(XR)](mod2″)

[ ]表示被[ ]包围的部分可以省略(以下同)，XR 省略时，表示不使用变址寄存器。

7．COMET 有 23 种指令，指令的名称，书写格式及其功能见附表 1，指令的书写格式用汇编语句的形式给出。

二、汇编语言 CASL 说明

COMET 的汇编语言是 CASL，其语法规则说明如下。

1．指令的种类和书写格式

CASL 由 4 种伪指令(START，END，DC，DC)3 种宏指令(IN，OUT，EXIT)和 23 种符号指令(COMET 的指令)组成。

CASL 的每条指令书写在一行内(最多不超过 72 个字符)．它的书写格式如附表 2 所示。

由附表 2 可知 CASL 每条指令由标号(可缺省)，指令码操作(可缺省)和注释(可缺省)4 栏构成，每一栏的书写规则如下：

标号栏：从第一个字符位里开始，最多不超过 6 个字符位置

指令码栏：在无标号时，从第二个字符位置以后的任意字符位置开始；在有标号时，

标号后面至少有一个空白，从其后的任意字符位置开始。

注释栏：行里有分号(；)时；其后直到终了作为注释处理(但 DC 指令里的字符串中的分号除外)。此外，在第一个字符位置为人号或在分号前只有空白的情况下，该行全部作为注释处理。在注释栏里，可以书写任何字符。

LABEL 泛指标号，标号最多不超过 6 个字符，开头必须是英文大写字母或数字。

用空白表示的栏目里不得写入字符。

附表 1

名 称	书 写 格 式	功 能
指令码		
操作数		
取数		
LD	GR,ADR[,XR]	
	(E)GR	
存数		
ST	GR,ADR[,XR]	
	(GR)E	
取地址*		
LEA	GR,ADR[,XR]	
	EGR	
加 法*		
ADD		

GR,ADR[,XR]

(GR)+(E)GR

减 法*

SUB

GR,ADR[,XR]

(GR)-(E)GR

逻辑乘*

AND

GR,ADR[,XR]

(GR)^(E)GR

逻辑加*

OR

GR,ADR[,XR]

(GR)v(E)GR

按位加*

EOR

GR,ADR[,XR]

(GR)(E)GR

算术比较

CPA

GR,ADR[,XR]

视(GR)和(E)为用补码表示的
带符号的二进制整数。
若(GR)＞E 则 00 FR

若(GR)＝(E)则 01 FR
若(GR)＜(E)则 10FR

逻辑比较

CPL

GR,ADR[,XR]

视(GR)和(E)为无符号的的
二进制非负整数。
若(GR)＞E 则 00 FR
若(GR)＝(E)则 01 FR
若(GR)＜(E)则 10FR

算术左移*

SLA

GR,ADR[,XR]

把(GR)向左或向右移动 E 位。在算术移位时，GR 中的第 0 位保持不变。在右移时空出的位被置成 0。在逻辑移位时,GR 中的 16 位一起移动，而空出的位被置成 0。

算术右移*

SRL

GR,ADR[,XR]

逻辑左移*

SLL

GR,ADR[,XR]

逻辑右移*

SRL

GR,ADR[,XR]

无条件转

JMP

ADR[,XR]

无条件向有效地址 E 转移，即 EPC

大于等于转

JPZ

ADR[,XR]

根据 FR 的值向有效地址 E 转移，即
EPC；
不转移时执行下一条指令，即
(PC)+2PC(注)

小于转

JMI

ADR[,XR]

不等转

JNZ

ADR[,XR]

等于转

JZE

ADR[,XR]

进 栈

PUSH

ADR[,XR]

(SP)-1SP,E(SP)

退 栈

POP

GR

((SP))GR,(SP)+1SP

调 用

CALL

ADR[,XR]

先形成有效地址 E,再依次执行
(SP)-1SP
(PC)+2(SP)
EPC

返 回

RET

((SP))PC,(SP)+1SP

注：

指 令

转移时 FR 的值

JPZ

00 或 01

JMI

10

JNZ

00 或 10

JZE

01

* 这些指令还根据执行结果(GR 中的值)，形成 FR 的值，其规则是：
若 GR 中的第 0 位为 0 且其各余各位不全为 0，则 FR 被置成 00；
若 GR 中的第 16 位全为 0，则 FR 被置成 01；
若 GR 中的第 0 位为 1，则 FR 被置成 10。

附录 2

标 号

指令码

操作数

注 释

[LABEL]

START

[LABEL]

空 白

END

空 白

[LABEL]

DC

常 数

[LABEL]

DS

区域的字数

[LABEL]

IN

ALABEL, NLABEL

[LABEL]

OUT

ALABEL, NLABEL

[LABEL]

EXIT

空　白

[LABEL]

空 白

2．伪指令

(1)[LABEL]START [LABEL]

表示程序的开头，即在程序的开始必须书写。

操作数栏中的标号是这个程序中定义的标号，它指出该程序的启动地址，在省略的情况下，程序从头开始执行。

标号栏中的标号可以作为其它的程序进人该程序的人口。

(2) END

表示程序的终止，在程序的末尾必须书。

(3)[LABEL］Dc 常数

用来指定和存储常数。常数分十进制常数，十六进制常数，地址常数和字符串常数四种。

标号栏中的标号是代表被指定的十进制常数、十六进制常数，地址常数的存贮地址或代表被指定的字符串常数的存储区域的第一个的地址。

十进制常数： DC n

用 n 指定一个十进制数（一 32768＜n＜65535），并将 n 转换成二进制数存储在一个字中，如果 u 超出规定的范围，则将其低 16 位再储起来。

对 32768～65535 的十进制数也可以用负的十进制常数表示。

十六进制常： DC ＃h

用 h 指定一个 4 位＋六进制数(0000 h FFFF). 并依次把字符数据存储在一个字中(在 h 的前面必需写上＃)。

地址常数： DC LABEL

将标号 LABEL 所对应的地址作为一个字的二进制数存储，若 LABEL 在该程序中没有定义，汇编将保留地址的定义，并由操作系统处理。

字符串常数： DC '字符串'

将字符串中从左开始的每个字符转换成字符数据（参阅上节 3），并依次把字符数据存储在连续的各字中。

在字符串中，不允许出现字符'；字符串的长度不能是零(即空字符串)

(4)［LABEL］DS 区域的字数

用来保图指定的字数的存储区域。

区域的字数，用十进制常数（≥0）指定。

标号栏中的标号是代表被保留的存储区域的第一个字的地址。

区域的字数为零时，存储区域不存在，但是标号栏中的标号仍有效,即代表下一字的地址。

3．宏指令

宏指令是根据事先定义的指令串和操作的信息，生成指定功能的指令串。

CASL 中有进行输人、输出及结束程序等宏指令，而没有定义输人、输出符号指令，这类处理由操作系统完成。

程序中出现没指令时，CASL 生成调用操作系统的指令串，但是，生成的指令中的字数不确定。

执行宏指令时，GR 的内容保持不变而 FR 的内容不确定。

(1)[LABEL] 1N ALAB2L，NLAB2L

宏指令 IN，从输人装置上输入一个记录，记录中的信息（字符）依次按字符数据的形式被顺序放在标号为 ALABEL 开始的区域内，已输入的字符个数以二进制数形式存放在标号为 ALABEL 的字中，记录之间的分配符号不输入。

(2)[LABEL] IN ALABEL，NLABEL

宏指令 OUT，将存放在从标号 ALABBL 开始的区域中的字符数据，作为一个记录向输出装置输出，输出的字符个数由标号为 NLABEL 的字中的内容指定，输出时，若要记录间的间隔符号，由操作系统自动插入输出。

(3)LABEL EXIT

宏指令 EXIT，表示程序执行的终止，控制返回操作系统。

4．符号指令，参阅上节 7。

5．指令字、区域的相对位置

由汇编生成的指令字和区域的相对位置按汇编语言程序的书写顺序确定，并且生成的指令字，区域占有主存的连续区域。

上午试题答案					
1	4	4	4	6	4

2	2	4	4	3	2
3	5	2	3	4	1
4	6	4	1	3	2
5	3	2	8	7	3
6	3	4	1	2	1
7	1	3	3	2	2
8	2	1	8	2	4
9	5	8	4	6	1
10	3	6	5	2	1
11	6	3	4	2	1
12	1	1	5	4	3
13	4	7	2	8	1

第一题		
	1	**收款员日销售文件：（每项 1 分）** **收款员号＋差错作业数＋差错金额＋退货次数＋退货金额** **商品文件：（每项 1 分）** **货号＋部门号＝单价** **部门日销售汇总文件：（每项 0.5 分，去除累计和的小数分）** **部门号＋货号＋销售数量＋销售总金额**
	2	**1、存在非法字符 或 非法数据 （如金额＜0）** **2、数量 X 单价 不等于 金额** **3、货号（或单价）与商品文件中的货号（或单价不一致**
	3	**删除其中的所有记录 或 初始化**
第二题		
	1	**判定覆盖：3**
	2	**条件覆盖：2**
	3	判定/条件覆盖：5
	4	条件组合覆盖：10
	5	路径覆盖：5
第三题		
	1	**J：0**

	2	**old[k]:"#"**
	3	**K+1->K**
	4	**old[k]:"#"**
	5	**K+1->K**
第四题		
	1	**ST GR0，NCH，GR2**
	2	**SLL GR2，8**
	3	**ADD GR2，NCH，GR3**
	4	**ST GR2，NCH，GR3**
	5	**LEA GR1，1，GR1 或 ADD GR1，C1**
第五题		
	1	**head->next**
	2	**q->next**
	3	**q->next**
	4	**p->next 答 q->next 给 1 分**
	5	**c--!=0 或 c-- 或 u!=v 或 i<=c 或 c-->0**
第七题		
	1	**sigma+d[i]**
	2	**sigma+d[i]**
	3	**flg[i]=0**
	4	**sigma**
	5	**s**

高级程序员计算机软件专业技术资格和水平考试

2000 年试题与答案

（2000 年度——上午试题）

试题 1

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

二叉树的前序、中序和后序遍历法最适合采用_A_来实现。

查找树中，由根结点到所有其他结点的路径长度的总和称为_B_，而使上述路径长度总和达到最小的树称为_C_，它一定是_D_。

在关于树的几个叙述中，只有_E_是正确的。

供选择的答案：

A： ①递归程序 ②迭代程序 ③队列操作 ④栈操作

B： ①路径和 ②内部路径长度 ③总深度 ④深度和

C： ①B-树 ②B+树 ③丰满树 ④穿线树
D： ①B-树 ②平衡树 ③非平衡树 ④穿线树
E： ①用指针方式存储有 n 个结点二叉树，至少要有 n+1 个指针
②m 阶 B-树中，每个非椰子结点的后件个数≥[m/2]
③m 阶 B-树中，具有 k 个后件的结点，必含有 k-1 个键值
④平衡树一定是丰满树

试题 2
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
当存储器采用段页式管理时，主存被划分为定长的_A_，程序按逻辑模块分成_B_。在某机器的多道程序环境下，每道程序还需要一个_C_作为用户标志号，每道程序都有对应的_D_。一个逻辑地址包括_C_x、段号 s、页号 p 和页内地址 d 四个部分。
设逻辑地址长度分配如下，其中 x、s、p、d 均以二进制数表示

21 20 19 14 13 11 10 0
x s p D
其转换后的物理地址为_E_。
供选择的答案：
A： ①段 ②页 ③区域 ④块
B： ①区域 ②页 ③块 ④段
C： ①模块号 ②区域号 ③基号 ④区域
D： ①一个段表和一个页表 ②一个段表和一组页表 ③一组段表和一个页表 ④一组段表和一组页表
E： ①x*220+s*214+p*211+d ②(((x)+s)+p)+d ③(((x)+s)+p)*211+(d)
④(((x)+s)+p)*211+d

试题 3
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
假设某程序语言的文法如下：
S→a|b|(T)
T→TdS|S
其中：VT={a，b，d，(,)},VN={S，T}，S 是开始符号。
考察该文法，称句型(Sd(T)db)是 S 的一个_A_，其中：_B_是句柄；_C_是素短语；_D_是该句型的直接短语；_E_是短语。
供选择的答案：
A： ①最左推导 ②最右推导 ③规范推导 ④推导
B、C： ①S ②b ③(T) ④Sd(T)
D： ①S ②S，(T)，b ③S，(T)，TdS，b ④(Sd(T)db)
E： ①(Sd(T)db) ②d(T) ③Td ④Sd(T)d

试题 4
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号

写在答卷的对应栏内。
_A_是操作系统中可以并行工作的基本单位，也是核心调度及资源分配的最小单位，它由_B_组成，它与程序的重要区别之一是：_C_。
在 SMP 系统中，操作系统还提供了_D_机制，它是_E_的最小单位。
供选择的答案：
A： ①作业 ②过程 ③函数 ④进程
B： ①程序、数据和标示符 ②程序、数据和 PCB ③程序、标示符和 PCB ④数据、标示符和 PCB
C： ①程序可占用资源，而它不可 ②程序有状态，而它没有
③它有状态，而程序没有 ④它能占有资源，而程序不能
D： ①约束 ②线程 ③共享 ④分时
E： ①存储器分配 ②资源分配 ③处理器分配 ④网络结点分配

试题 5
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
在软件开发过程中常用图作为描述工具。DFD 就是面向_A_分析方法的描述工具。在一套分层 DFD 中，如果某一张图中有 N 个加工（process），则这张图允许有_B_张子图。在一张 DFD 图中，任意两个加工之间_C_。在画分层 DFD 时，应注意保持_D_之间的平衡。DFD 中从系统的输入流到系统的输出流的一连串连续变换形成一种信息流，这种信息流可分为_E_两大类。
供选择的答案：
A： ①数据结构 ②数据流 ③对象 ④构件（component）
B： ①0 ②1 ③1～N ④0～N
C：①有且仅有一条数据流 ②至少有一条数据流 ③可以有 0 或多条名字互不相同的数据流
④可以有 0 或多条数据流，但允许其中有若干条名字相同的数据流
D： ①父图与子图 ②同一父图的所有子图 ③不同父图的所有子图 ④同一子图的所有直接父图
E： ①控制流和变换流 ②变换流和事务流 ③事务流和事件流 ④事件流和控制流

试题 6
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
软件的易维护性是指理解、改正、改进软件的难易度。通常影响软件易维护性的因素有易理解性、易修改性和_A_。在软件的开发过程中往往采取各种措施来提高软件的易维护性。如采用_B_有助于提高软件的易理解性；_C_有助于提高软件的易修改性。
在软件质量特性中，_D_是指在规定的一段时间和条件下，与软件维持其性能水平的能力有关的一组属性；_E_是指防止对程序及数据的非授权访问的能力。
供选择的答案：
A： ①易使用性 ②易恢复性 ③易替换性 ④易测试性
B： ①增强健壮性 ②信息隐蔽原则 ③良好的编程风格 ④高效的算法

C： ①高效的算法 ②信息隐蔽原则 ③增强健壮性 ④身份认证
D： ①正确性 ②准确性 ③可靠性 ④易使用性
E： ①安全性 ②适应性 ③灵活性 ④容错性

试题 7
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
域表达式{ab|R(ab)∧R(ba)}转换成为等价的关系代数表达式，所列出的式子中_A_是不正确的。SQL 中集合成员资格的比较操作"元组 NOT IN（集合）"中的"NOT IN"与_B_操作符等价。SQL 中涉及属性 AGE 是否是空值的比较操作，写法_C_是错误的。类似于"工资在 800 至 5000 之间"这种约束，是属于 DBS 的_D_功能。设关系模式 Q 是 3NF 模式，那么，_E_这种提法是不正确的。
供选择的答案：
A： ① ②
③ ④
B： ①<>SOME ②=SOME ③<>ALL ④=ALL
C： ①AGE IS NULL ②NOT(AGE IN NULL) ③AGE=NULL ④AGE IS NOT NULL
D： ①完整性 ②并发控制 ③安全性 ④恢复
E： ①Q 一定是 2NF 模式 ②Q 可能不是 4NF 模式 ③Q 可能不是 BCNF ④Q 一定不是 BCNF

试题 8
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
数据压缩技术是多媒体信息处理中的关键技术之一，数据压缩技术可分为_A_两大类。_B_是一种与频度相关的压缩编码方法，_C_主要用于视频信息的压缩，_D_常用于静止图片的信息压缩。由三基色（RGB）原理出发的 RGB 彩色空间，在多媒体技术中是最常用的，此外还有多种彩色空间，但_E_不是计算机上用的彩色空间。
供选择的答案：
A： ①可逆与不可逆 ②高速与低速 ③编码与非编码 ④冗余与非冗余
B： ①MIPS ②ISDN ③Huffman ④Gauss
C、D： ①MIPS ②MPEG ③JPEG ④JIPS
E： ①YUV ②HIS ③XYZ ④IMG

试题 9
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
直接存储器访问（DMA）是一种快速传递大量数据常用的技术。工作过程大致如下：
（１）向 CPU 申请 DMA 传送；
（２）获 CPU 允许后，DMA 控制器接管_A_的控制权；
（３）在 DMA 控制器的控制下，在存储器和_B_之间直接进行数据传送，在传送过程中不需要_C_的参与。开始时需提供要传送的数据的_D_和_E_。

（4）传送结束后，向 CPU 返回 DMA 操作完成信号。
供选择的答案：
A： ①系统控制台 ②系统总线 ③I/O 控制器 ④中央处理器
B： ①外部设备 ②运算器 ③缓存 ④中央处理器
C： ①外部设备 ②系统时钟 ③系统总线 ④中央处理器
D： ①结束地址 ②起始地址 ③设备类型 ④数据速率
E： ①结束地址 ②设备类型 ③数据长度 ④数据速率

试题 10
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
假设某计算机具有 1M 字节的内存（目前使用的计算机往往具有 64M 字节以上的内存），并按字节编址，为了能存取该内存各地址的内容，其地址寄存器至少需要二进制_A_位。为使 4 字节组成的字能从存储器中一次读出，要求存放在存储器中的字边界对齐，一个字的地址码应_B_。若存储周期为 200NS，且每个周期可访问 4 个字节，则该存储器带宽为_C_BIT/S。假如程序员可用的存储空间为 4M 字节，则程序员所用的地址为_D_，而真正访问内存的地址称为_E_。
供选择的答案：
A： ①10 ②16 ③20 ④32
B： ①最低两位为 00 ②最低两位为 10 ③最高两位为 00 ④最高两位为 10
C： ①20M ②40M ③80M ④160M
D： ①有效地址 ②程序地址 ③逻辑地址 ④物理地址
E： ①指令 ②物理地址 ③内存地址 ④数据地址

试题 11
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
现采用四级流水线结构分别完成一条指令的取指、指令译码和取数、运算以及送回运算结果四个基本操作，每步操作时间依次为 60ns，100ns，50ns 和 70ns。该流水线的操作周期应为_A_ns。若有一小段程序需要用 20 条基本指令完成（这些指令完全适合于流水线上执行），则得到第一条指令结果需_B_ns，完成该段程序需_C_ns。
在流水线结构的计算机中，频繁执行_D_指令时会严重影响机器的效率。当有中断请求发生时，采用不精确断点法，则将_E_。
供选择的答案：
A： ①50 ②70 ③100 ④280
B： ①100 ②200 ③280 ④400
C： ①1400 ②2000 ③2300 ④2600
D： ①条件转移 ②无条件转移 ③算术运算 ④访问存储器
E： ①仅影响中断反应时间，不影响程序的正确执行 ②不仅影响中断反应时间，还影响程序的正确执行
③不影响中断反应时间，但影响程序的正确执行 ④不影响中断反应时间，也不影响程序的正确执行

试题 12
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
为提高数据传输的可靠性，可采用"冗余校验"的方法。海明码是常用的方法之一。在此方法中，若要求能检测出所有的双位错，并能校正单位错，则合法码字集中的码距至少为_A_。若原始数据的字长为 5 位，则采用海明码时其校验位至少为_B_位。
对下面图(a)所是系统，仅当部件 1，部件 2 和部件 3 全部正常工作时系统才能正常工作。图中数字为各部件的可靠性，整个系统的可靠性近似为_C_。如果将部件 2 和部件 3 改成由两个器件构成，如图(b)所示，只要器件 a 和 b 中有一个正常就能使部件 2 正常工作，只要器件 c 和 d 中有一个正常就能使部件 3 正常工作。图中数字是各器件可靠性，则部件 2 的可靠性是_D_，整个系统的可靠性近似为_E_。

器件 a 器件 c

部件 1 部件 2 部件 3 部件 1

器件 b 器件 d
部件 2 部件 3
图（a） 图（b）
供选择的答案：
A、B： ①1 ②2 ③3 ④4
C： ①0.68 ②0.72 ③0.80 ④0.92
D： ①0.64 ②0.88 ③0.96 ④0.99
E： ①0.82 ②0.90 ③0.94 ④0.96

试题 13
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。
计算机网络中对等实体间通信时必须遵循约定的标准协议。不同的协议有不同的功能，如：
_A_：用于网络管理；
_B_：用于移动电话无线上网；
_C_：可用于家庭拨号上网；
_D_：一种面向比特的数据链路通信规程；
_E_：一种路由选择协议。
供选择的答案：
A～C： ①SNMP ②PPP ③RIP ④WAP
D、E： ①OSPF ②HTTP ③HDLC ④RARP

试题 14
从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

Network managers have long _A_ practical voice-over-IP(VOIP) solutions. VOIP _B_ ease network management and decreases costs by converging a company's telephony and data infrastructures into one network. And a VOIP solution implemented at a company's head-quarters with far-reaching branch offices can _C_ tremendous amounts of _D_ in long distance phone bills, provided that solution delivers POTS-like voice _E_ over the Internet.

供选择的答案：

A: ①await ②awaited ③awaiting ④awaits

B: ①promise ②promised ③promises ④promising

C: ①get ②put ③save ④waste

D: ①cash ②money ③space ④time

E: ①frequency ②length ③quality ④quantity

试题 15

从供选择的答案中，选出应填入下面叙述中_?_内的最确切的解答，把相应编号写在答卷的对应栏内。

Basically, a firewall is a _A_ process or a set of integrated processes that runs on a router or _B_ to control the flow of networked application traffic _C_ through it. Typically, firewalls are placed on the entry point to a _D_ network such as the Internet. They could be considered traffic cops. The firewall's _E_ is to ensure that all communication between an organization's network and the Internet conform to the organization's security policies.

供选择的答案：

A: ①dependent ②isolated ③related ④standalone

B: ①browser ②client ③repeater ④sever

C: ①pass ②passed ③passing ④passes

D: ①local ②national ③private ④public

E: ①mole ②pole ③role ④rule

（2000 年度——下午试题）

从以下的 3 道试题（试题一至试题三）中任选 2 道解答。如果解答的试题数超过 2 道，则解答的前 2 道有效。

试题一

阅读以下说明和流程图，回答问题 1 和问题 2，将答案写在答卷的对应栏内。

【流程图】

【说明】

本流程图实现从成绩文件生成学生成绩一览表。

某中学某年级的学生成绩数据（分数）登录在成绩文件 F0 中，其纪录格式如下：

学号 姓名 课程 1 成绩 课程 2 成绩 …… 课程 6 成绩

由该成绩文件生成如下表所示的学生成绩一览表。生成的学生成绩一览表按学号升序排列。表中的名次是指该生相应课程在年级中的名次。

学号 姓名 课程 1 课程 2 …… 课程 6
成绩 名次 成绩 名次 …… …… 成绩 名次

流程图中的顺序文件 F0 是学生成绩文件，F0 文件经处理 1 处理后产生顺序文件 F，然后经过处理 2 至处理 4 对文件 F 进行处理和更新。在处理 5 中，仅对文件 F 的纪录进行学生成绩一览表的编排输出，不进行排序和增加名次等处理。
［问题 1 ］
流程图中文件 F 的纪录格式设定为如下形式：

学号 姓名 课程代码 ① ②
其中的①、②应定义为何种数据项？
［问题 2 ］
简述处理 2、处理 3 和处理 4 作何种处理，若有排序处理则需指明排序的键及序（升序或降序）。

试题二
阅读以下说明和流程图，回答问题 1 至问题 4，将答案写在答卷的对应栏内。
【说明】
本流程图是将中缀表示的算术表达式转换成后缀表示。如中缀表达式
(A-(B*C+D)*E)/(F+G)
的后缀表示为
ABC*D+E*-FG+/
为了方便，假定变量名为单个英文字母，运算符只有+、-、*、/（均为双目运算符，左结合），并假定所提供的算术表达是非空且语法是正确的。另外，中缀表示形式中无空格符，但整个算术表达式以空格符结束。流程图中使用的符号的意义如下：
数组 IN[] 存储中缀表达式；
数组 POLISH[] 存储其后缀表达式；
数组 S[] 是一个后进先出栈；
函数 PRIOR(CHAR) 返回符号 CHAR 的优先级，各符号的优先级如下表所示：CHAR PRIOR(CHAR)
* /+ -() 4321

［问题 1 ］
填充流程图中①的判断条件。
［问题 2 ］
写出子程序 A 的功能，并顺序写出实现该功能的操作。
［问题 3］
写出子程序 B 的功能，并顺序写出实现该功能的操作。
［问题 4］
中缀表达式
(A+B-C*D)*(E-F)/G

经该流程图处理后的输出是什么？
【流程图】

试题三
阅读以下说明和流程图，回答问题 1 和问题 2，将答案写在答卷的对应栏内。
【说明】
某供销系统接受顾客的订单，当库存中某配件的数量小于定购量或库存量低于一定数量时，向供应商发出采购单；当某配件的库存量大于或等于定购粮食，或者收到供应商的送货单并更新了库存后，向顾客发出提货单。该系统还可随时向总经理提供销售和库存情况表。该供销系统的分层数据流图中部分数据流和文件的组成如下：
文件
配件库存=配件号+配件名+规格+数量+允许的最低库存量
数据流
订货单=配件号+配件名+规格+数量+顾客名+地址
提货单=订货单+金额
采购单=配件号+配件名+规格+数量+供应商名+地址
送货单=配件号+配件名+规格+数量+金额
假定顶层图是正确的，"供应商"文件已由其他系统生成。
［问题 1 ］
指出哪张图中的哪些文件可不必画出。
［问题 2 ］
指出在哪些图中遗漏了哪些数据流。回答时使用如下形式之一：
（ 1 ）XX 图中遗漏了 XX 加工（或文件）流向 XX 加工（或文件）的 XX 数据流；
（ 2 ）XX 图中 XX 加工遗漏了 XX 输入（或输出）数据流。
【流程图】
顶层图

订货单 送货单
不合法订货单 不合法送货单
提货单 采购单

查询销售及库存情况 销售及库存情况

0 层图

订货单 缺货订单 采购单

不合格订货单 送货单

采购请求

销售及库存情况 不合格送货单
到货通知

查询销售及库存情况

提货单 配件库存 采购清单

加工 1 子图

订货单 合格订货单 采购请求

不合格订货单

有货
配件库存 订单 缺货订单

销售及库存情况表 到货单

查询销售及库存情况

销售历史

加工 2 子图

供应商

按配件汇总的采购单 采购单

缺货清单 采购清单
需增配件清单 不合格送货单

送货单

到货通知

配件库存

下列 3 道试题（试题四至试题六）为必答题。
试题四
在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读程序说明和

CASL 程序，将应填入_(n)_处的字句，写在答卷的对应栏内。
【程序 4 说明】
①本子程序根据每位职工的基本工资（非负值）和他完成产品的超额数或不足数计算该职工的应发工资。
②主程序调用时，GR1 中给出子程序所需参数的起始地址，参数的存放次序如下表：

GR1→ a1
b1
c1
a2
b2
c2
⋮
an
bn
cn
-1（结束标志）

其中：ai 为职工 i 的基本工资；bi 为职工 i 的完成产品的超额数或不足数；ci 为职工 i 的应发工资数（i=1，2，…，n）。
bi 以原码形式存放（大于零为超额，小于零为不足），基本工资与计算所得的应发工资以补码形式存放。
③应发工资的计算规则为：
o 恰好完成定额数（此时 bi 为零），应发工资即为基本工资。
o 每超额 4 件，在基本工资基础上增加 10 元（不到 4 个，以 4 计算。例如超额数为 10 时，增加 30 元）。
o 每不足 4 件，在基本工资基础上减 5 元（不到 4 个，以 4 计算。例如，不足数为 5 时，减 10 元）。
【程序 4】

```
START
BEG PUSH 0,GR1
PUSH 0,GR2
PUSH 0,GR3
L1 _(1)_
LEA GR0,0,GR2
JMI FINISH
LD GR3,1,GR1
LEA GR2,0,GR3
AND GR2,C7FFF
JNE L3
SRL GR3,15
LEA GR2,-1,GR2
L2 _(2)_
```

```
LEA GR2,-4,GR2
JPZ L2
L3 _(3)_
_(4)_
_(5)_
FINISH POP GR3
POP GR2
POP GR1
RET
C7FFF DC #7FFF
BONUS DC 10
DC -5
END
```

试题五

阅读下列程序说明和C代码，将应填入_(n)_出的字句写在答卷的对应栏内。

【程序5说明】

下列文法可用来描述化学分子式的书写规则（例如，A12(CO3)3、Cu(OH)2）：

λ → β | β λ

β → δ | δ n

δ → ξ | ξ θ | (λ)

其中：λ 是一个分子式；δ 或是一个元素，或是一个带括号的（子）分子式，元素或是一个大写字母（记为 ξ），或是一个大写字母和一个小写字母（记为 ξ θ）；β 或是一个 δ，或是在 δ 之后接上一个整数 n，δ n 表示 β 有 n 个 δ 的元素或（子）分子式。一个完整的分子式由若干个 β 组成。

当然一个正确的分子式除符合上述文法规则外，还应满足分子式本身的定义要求。

下面的程序输入分子式，按上述文法分析分子式，并计算出该分子式的分子量。例如：元素H的原子量是1，元素O的原子量是16。输入分子式H2O，程序计算出它的分子量为18(1*2+16)。程序中各元素的名及它的原子量从文件atom.dat中读入。

【程序5】

```
#include <stdio.h>
#include <string.h>
#define MAXN 300
#define CMLEN 30
struct elem {char name[]; /*元素名*/
double v; /*原子量*/
}nTbl[MAXN];
char cmStr[CMLEN],*pos;
int c; FILE *fp;
double factor();
double atom() /*处理文法符号 δ*/
```

```
{char w[3]; int i; double num;
while((c=*pos++)==''||c=='\t'); /*略过空白字符*/
if(c=='\n') return 0.0;
if(c>='A'&&c<='Z'){ /*将元素名存入 W*/
w[i=0]=c; c=*pos++;
if(c>='a'&&c<='z') w[++i]=c; else pos--;
w[++i]='\0';
for(i=0;nTbl[i].v>0.0;i++)
if(strcmp(w,nTbl[i].name)==0) return nTbl[i].v;
printf("\n 元素表中没有所输入的元素：\t%s%n",w); return -1.0;
}elseif(c=='('){
if((num=_(1)_)<0.0) return -1.0; /*包括可能为空的情况*/
if(*pos++!=')'){printf("分子式中括号不匹配！\n"); return -1.0;}
return num;
}
double mAtom() /*处理文法符号 β*/
{double num; int n=1;
if((num=_(2)_)<0.0) return -1.0;
c=*pos++;
if(c>='0'&&c<='9'){
n=0; while(c>='0'&&c<='9')
{n=_(3)_;
c=*poss++;
}
}
pos--;
return num*n;
}
double factor() /*处理文法符号 λ*/
{double num=0.0,d;
if((num=mAtom())<.0.0) return -1.0;
while(*pos>='A'&&*pos<='Z'||*pos=='('){
if((d=_(4)_)<0.0) return -1.0;
_(5)_;
}return num;
}
void main()
{char fname[]="atom.dat"; /*元素名及其原子量文件*/
int i; double num;
if((fp=fopen(fname,"r"))==NULL){ /*以读方式打开正文文件*/
printf("Can not open %s file. \n",fname); return;
}
i=0;
while(i<MAXN&&fscanf(fp,"%s%lf",bTbl[i].name,&nTbl[i].v)==2)
```

```
i++;
fclose(fp); nTbl[i].v=-1.0;
while(1){ /*输入分子式和计算分子量循环，直至输入空行结束*/
printf("\n 输入分子式！（空行结束）\n"); gets(cmStr);
pos=cmStr;
if(cmStr[0]=='\0') break;
if((num=fator())>0.0)
if(*pos!='\0') printf("分子式不完整！\n");
else printf("分子式的分子量为%f\n",num);
}
}
```

试题六

阅读下列程序说明和 C 代码，将应填入_(n)_出的字句写在答卷的对应栏内。

【程序 6 说明】

设某城市有 n 个车站，并有 m 条公交线路连接这些车站，设这些公交车都是单向的，这 n 个车站被顺序编号为 0 至 n-1。本程序，输入该城市的公交线路数、车站个数，以及各公交线路上的各站编号，求得从站 0 出发乘公交车至站 n-1 的最少换车次数。

程序利用输入信息构建一张有向图 G（用邻接矩阵 g 表示），有向图的顶点是车站，若有某条公交线路经 i 站能到达 j 站，就在顶点 i 到顶点 j 之间设置一条权为 1 的有向边<i，j>。如是这样，从站点 x 至站点 y 的最少上车次数便对应图 G 中从点 x 至点 y 的最短路径长度。而程序要求的换车次数就是上车次数减 1。

【程序 6】

```
#include <stdio.h>
#define M 20
#define N 50
int a[N+1]; /*用于存放一条线路上的各站编号*/
int g[N][N]; /*存储对应的邻接矩阵*/
int dist[N]; /*存储站 0 到各站的最短路径*/
int m,n;
void buildG()
{int I,j,k,sc,dd;
printf("输入公交线路数，公交站数\n");
for(i=0;i<n;i++) /*邻接矩阵清 0*/
for(j=0;j<n;j++) g[i][j]=0;
for(i=0;i<m;i++){
printf("沿第%d 条公交车线路前进方向的各站编号（0<=编号<=%d，-1 结束）
",i+1,n-1);
sc=0; /*当前线路站计数器*/
while(1){
scanf("%d",&dd);
if(dd==-1) break;
if(dd>=0&&dd<n) _(1)_;
```

```
}
a[sc]=-1;
for(k=1;a[k]>=0;k++) /*处理第 i+1 条公交线路*/
for(j=0;j<k;j++)
g_(2)_=1;
}
}
int minLen()
{int j,k;
for(j=0;j<n;j++) dist[j]=g[0][j];
dist[0]=1;
do{ for(k=-1,j=0;j,n;j++) /*找下一个最少上车次数的站*/
if(dist[j]>0&&(k==-1||dist[j]<dist[k])) k=j;
if(k<0||k==n-1) break;
dist[k]=-dist[k]; /*设置 k 站已求得上车次数的标记*/
for(j=1;j<n;j++) /*调节经过 k 站能到达的其余各站的上车次数*/
if(_(3)_&&(dist[j]==0||-dist[k]+1<dist[j]))
dist[j]=_(4)_;
}while(1);
j=dist[n-1];
return _(5)_;
}
void main()
{int t;
buildG();
if((t=minLen())<0) printf("无解! \n");
else printf("从 0 号站到%d 站需换车%d 次\n",n-1,t);
}
```

（2000 年高程答案）

上午试题答案

	A	B	C	D	E
1	1	2	3	2	3
2	2	4	3	2	4
3	4	1	3	2	1
4	4	2	3	2	3
5	2	4	3	1	2
6	4	3	2	3	1
7	2	3	3	1	4
8	1	3	2	3	4
9	2	1	4	2	3
10	3	1	4	3	2
11	3	4	3	1	2
12	3	4	1	3	2
13	1	4	2	3	1
14	2	3	3	2	3
15	4	4	3	4	3

下午试题答案

试题一

［问题 1 ］

①成绩 ②名次

［问题 2 ］

按课程代码（升序）、成绩（降序）

处理 2：对每个文件 F 排序

处理 3：对每个课程代码，决定学生名次，写入文件 F 的相应字段

处理 4：按学号（升序）、课程代码（降序）对文件 F 排序

试题二

［问题 1 ］

PRIOR（IN[i]）：PRIOR（S[p]）

［问题 2 ］

功能： 将当前符号 IN[i]入栈

操作： p+1->p

IN[i]->S[p]

［问题 3］

功能： 出栈

操作： k+1->k

S[p]->POLISH[k]

p-1->p

［问题 3］

AB+CD*-EF-*G/

试题三

［问题 1 ］

0 层图中的"采购清单"多余应去掉

［问题 2］
加工 1 子图中遗漏了"配件库存"文件到 1.3 加工的数据流
加工 1 子图中 1.4 加工遗漏了"提货单"输出数据流
加工 1 子图中 1.5 加工遗漏了"到货通知"输入数据流
加工 2 子图中 2.3 加工遗漏了"采购请求"输入数据流

试题四
（1） LD GR2,0,GR1
（2） ADD GR0,BONUS,GR3
（3） ST GR0,2,GR1
（4） LEA GR1,3,GR1
（5） JMP L1

试题五
（1） factor()
（2） atom(0
（3） n*10+c-'10'
（4） mAtom()
（5） num+=d

试题六
（1） a[sc++]=dd
（2） a[a[j]][a[k]]
（3） （dist[j]>=0）&&（g[k][j]==1）
（4） -dist[k]+1
（5） k<0 ? -1 : j-1

高级程序员计算机软件专业技术资格和水平考试

2001 年试题与答案

（2001 年度——上午试题）

一棵查找二叉树,其结点 A、B、C、D、E、F 依次存放在一个起始地址为 n（ 假定地址以字节为单位顺序编号 ）的连续区域中,每个结点占 4 个字节:前二个字节存放结点值,后二个字节依次放左指针、右指针。
若该查找二叉树的根结点为 E ,则它的一种可能的前序遍历为__(1)__ ,相应的层次遍历为__(2)__。在以上两种遍历情况下,结点 C 的左指针 Lc 的存放地址为__(3)__,Lc 的内容为__(4)__。结点 A 的右指针 RA 的内容为__(5)__。
(1):A.EAFCBD B.EFACDB C.EABCFD D.EACBDF
(2):A.EAFCBD B.EFACDB C.EABCFD D.EACBDF

(3):A.n+9 B.n+10 C.n+12 D.n+13
(4):A.n+4 B.n+8 C.n+12 D.n+16
(5):A.n+4 B.n+8 C.n+12 D.n+16

用来辅助软件开发、运行、维护、管理、支持等过程中的活动的软件称为软件开发工具,通常也称为__(6)__工具。
(6):A.CAD B.CAI C.CAM D.CASE

使用软件开发工具有助于提高软件的开发、维护和管理的效率。集成型软件开发环境通常由工具集和环境集成机制组成。这种环境应具有__(7)__。环境集成机制主要有数据集成机制、控制集成机制和界面集成机制。
(7):A.开放性和可剪裁性 B.开放性和不可剪裁性
C.封闭性和可剪裁性 D.封闭性和不可剪裁性

·数据集成机制为环境中的工具提供统一的__(8)__；控制集成机制为__(9)__提供支持。
界面集成机制使得环境中的所有工具具有__(10)__。
(8):A.数据类型 B.数据模式 C.数据接口规范 D.数据仓库
(9):A.使各工具具有统一的控制结构 B.备工具之间的通信、切换、调度和协同工作
C.使备工具具有统一的控制接口 D.各工具之间的同步开发
(10):A.同一个界面 B.相同的图标和图标的含义
C.统一的界丽风格相操作方式 D.相同个数的窗口和菜单项

递归算法的执行过程,一般来说,可先后分成__(11)__和__(12)__两个阶段。
(11):A.试探 B.递推 C.枚举 D.分析
(12):A.回溯 B.回归 C.返回 D.合成

若一个问题的求解既可以用递归算法,也可以用递推算法,则往往用__(13)__算法,因为__(14)__。
(13):A.先递归后递推 B.先递推后递归 C.递归 D.递推
(14):A.递推的效率比递归高 B.递归宜于问题分解
C.递归的效率比递推高 D.递推宜于问题分解

贪婪法是一种__(15)__的算法。
(15):A.不求最优,只求满意 B.只求最优 C.求取全部可行解 D.求取全部最优解

考察下列文法: G(VT ,VN ,E ,P)
其中: VT = { + , * ,(,) , i }
VN = { E , T , F }
E 是开始符号
P:
E → E + T | T
T → T * F | F
F → (E)| i
F*F+T 是该文法的一个句型,其中 ,__(16)__是句柄 ,__(17)__是素短语。__(18)__是该句型的

直接推导,__(19)__是该句型的最左推导。__(20)__是该文法的一个句子。
(16):A.F B.F*F C.F+T D.F*F+,T
(17):A.F B.F*F C.F+T D.F*F+T
(18):A.F*F+i B.F*F+T*F C.F*F+F*F D.i*i+T
(19):A.F*F+T*F B.F*F+T C.F*(E)+T D.(E)*F+T
(20):A.T+(i+i) B.i+(i+F) C.i D.(E)

软件开发模型用于指导软件的开发。演化模型是在快速开发一个__(21)__的基础上,逐步演化成最终的软件。
(21):A.模块 B.运行平台 C.原型 D.主程序

螺旋模型综合了__(22)__的优点,并增加了__(23)__。
(22):A.瀑布模型和演化模型 B.瀑布模型和喷泉模型
C.演化模型和喷泉模型 D.原型和喷泉模型
(23):A.质量评价 B.进度控制 C.版本控制 D.风险分析

喷泉模型描述的是面向__(24)__的开发过程,反映了该开发过程的__(25)__特征。
(24):A.数据流 B.数据结构 C.对象 D.构件(component)
(25):A.迭代和有间隙 B.迭代和无间隙 C.无迭代和有间隙 D.无迭代和无间隙

MIDI 是一种数字音乐的国际标准,MIDI 文件存储的__(26)__。它的重要特色是__(27)__。
(26):A.不是乐谱而是波形 B.不是波形而是指令序列
C.不是指令序列而是波形 D.不是指令序列而是乐谱
(27):A.占用的存储空间少 B.乐曲的失真度少 C.读写速度快 D.修改方便

MPEG 是一种__(28)__,它能够__(29)__。
(28):A.静止图像的存储标准 B.音频、视频的压缩标准
C.动态图像的传输标准 D.图形国家传输标准
(29):A.快速读写 B.有高达 200:1 的压缩比
C.无失真地传输视频信号 D.提供大量基本模板

MPEG-4 相对于 MPEG 的以前版本的最大特点是__(30)__。
(30):A.更高的压缩比 B.更多的基本图形的模板
C.更强的交互能力 D.更快的运算速度

关系数据库设计理论主要包括三个方面内容,其中起核心作用的是__(31)__。
(31):A.范式 B.数据模式 C.数据依赖 D.范式和数据依赖

在关系模型的完整性约束中,实体完整性规则是指关系中__(32)__。引用完整性规则要求__(33)__。
(32):A.不允许有主行 B.属性值不允许为空
C.主键值不允许为空 D.外键值不允许为空
(33):A.不允许引用不存在的元组 B.允许引用不存在的元组
C.不允许引用不存在的属性 D.允许引用不存在的属性

设关系 P 和 Q 分别为 2 元和 3 元关系,则与关系代数表达式 P Q 等价的是__(30)__。
(34):A.σ1<2(P×Q) B.σ1<4(P×Q) C.σ1<2(P Q) D.σ1<4(P Q)
对于基本表 S(S#,NAME,SEX,BIRTEBAY)和 SC(S#,C#,GRADE),其中 S#、NAME、SEX、BIRTHDAY、C#和 GRADE 分别表示学号、姓名、性别、生日、课程号和成绩。
有一 SQL 语句:

```
SELECT S# NAME
FROM S
WHERE S# NOT IN
( SELECT S#
FROM S
WHERE C# = 'c102');
```

其等价的关系代数表达式是__(35)__。
(35):A.∏s#,name(σc#≠ 'c102'(S SC))
B.∏s#,name(S)-∏s#,name (σc#= 'c102' (S SC))
C.∏s#,name(S σc# ≠ 'c102' (SC))
D.∏s#name(S SC)

因争用资源产生死锁的必要条件是互斥、循环等待、不可抢占和__(36)__。对于缓冲池(大量缓冲区)的管理,采用生产者-消费者方式解决同步或互斥时,通常需要用__(37)__个信号量。
(36):A.请求与释放 B.释放与保持 C.释放与阻塞 D.保持与等待。
(37):A.1 B.2 C.3 D.4

虚存页面调度算法有多种,__(38)__调度算法不是页面调度算法。
(38):A.后进先出 B.先进先出 C.最近最少使用 D.随机选择

实存的存储分配算法用来决定输入的程序和数据放到主存中的位置,采用"总是把程序装入主存中最大的空闲区域"的算法称为__(39)__。
(39):A.最佳适应算法 B.最差适应算法 C.首次适应算法 D.循环首次适应算法

在文件存储设备管理中,有三类常用的空闲块管理方法,即位图向量法,空闲块链表链接法和__(40)__。
(40):A.一级目录法 B.多级自录法 C.分区法 D.索引法

用二进制加法器对二一十进制编码的十进制数求和,当和的本位十进制数二一十进制编码小于等于 1001 且向高位无进位时,__(41)__;当和小于等于 100l 且向高位有进位时,__(42)__;当和大于 1001 时,__(43)__。
(41)～(43):A.不需进行修正 B.需进行加 6 修正
C.需进行减 6 修正 D.进行加 6 或减 6 修正,需进一步判别

微机用的系统,总线有多种,其中__(44)__是早期工业标准体系结构单总线的直接扩展,并向下与其兼容;__(45)__首先应用于奔腾机,数据线宽度已可为 64 位,并有数据缓冲能力。
(44)、(45):A. VESA B.SCSI C.EISA D.PCI

被操作数的最高位移入"进位",位,其余所有位接收其相邻低位值,最低位移入 0 的操作是__(46)__指令。被操作数的最高位保持不变,其余所有位接收其相邻高位值,最低位移到"进位"位中的操作是__(47)__指令。在程序执行过程中改变按程序计数器顺序读出指令的指令属于__(48)__。相对寻址方式的实际地址是__(49)__。特权指令在多用户、多任务的计算机系统中必不可少,它主要用于__(50)__。

(46)、(47):A.逻辑左移 B.算术左移 C.乘 2 运算 D.除 2 运算

(48):A.特权指令 B.传送指令 C.输入输出指令 D.转移指令

(49):A.程序计数器的内容加上指令中形式地址值

B.基值寄存器的内容加上指令中形式地址值

C.指令中形式地址中的内容 D.栈顶内容。

(50):A.检查用户的权限 B.系统硬件自检和配置

C.用户写汇编程序时调用 D.系统资源的分配和管理

利用并行处理技术可以缩短计算机的处理时间,所谓并行性是指__(51)__。可以采用多种措施来提高计算机系统的并行性,它们可分成三类,即__(52)__。

提供专门用途的一类并行处理机(亦称阵列处理机)以__(53)__方式工作,它适用于__(54)__。多处理机是目前较高性能计算机的基本结构,它的并行任务的派生是__(55)__。

(50):A.多道程序工作 B.多用户工作

C.非单指令流单数据流方式工作 D.在同一时间完成两种或两种以上工作

(52):A.多处理机,多级存储器和互连网络 B.流水结构,高速缓存和精简指令集

C.微指令,虚拟存储和 I/O 通道 D.资源重复,资源共享和时间重叠。

(53):A.SISD B.SIMD C.MISD D.MB4D

(54):A.事务处理 B.工业控制 C.矩阵运算 D.大量浮点计算

(55):A.需要专门的指令来表示程序中并发关系和控制并发执行

B.靠指令本身就可以启动多个处理单元并行工作

C.只执行没有并发约束关系的程序

D.先并行执行,事后再用专门程序去解决并发约束

发展容错技术可提高计算机系统的可靠性。利用元件冗余可保证在局部有故障情况下系统的正常工作。带有热备份的系统称为__(56)__系统。它是__(57)__,因此只要有一个子系统能正常工作,整个系统仍能正常工作。

当子系统只能处于正常工作和不工作两种状态时,我们可以采用图 A 的并联模型,若单个子系统的可靠性都为 0.8 时,图示的三个子系统并联后的系统可靠性为__(58)__。若子系统能处于正常和不正常状态时,我们可以采用图 B 所示的表决模型,若图中有任何二个或三个子系统输出相同时,则选择该相同的输出作为系统输出,设单个子系统的可靠性为 0.8 时,整个系统的可靠性为__(59)__;若单个子系统的可靠性为 0.5 时,整个系统的可靠性为__(60)__。

[图 A] [图 B]

(56):A.并发 B.双工 C.双重 D.并行。

(57):A.两子系统同时同步运行,当联机子系统出错时,它退出服务,由备份系统接替

B.备份系统处于电源开机状态,一旦联机子系统出错时,立即切换到备份系统

C.两子系统交替处于工作和自检状态,当发现一子系统出错时,它不再交替到工作状态

D.两子系统并行工作,提高机器速度,一旦一个子系统出错,放弃并行工作。

(58):A.0.9 B.O.94 C.O.992 D.O.996

(59):A.0.882 B.O.896 C.O.925 D.O.94
(60):A.0.5 B.O.54 C.O.62 D.O.65

WWW 页面访问的大致过程如下:用户在浏览器中输入要访问的 WWW 页面的__(61)__地址(http://hostnane/dirctory/fi1e);浏览器通过__(62)__查询上述输入信息所指的 Web 服务器的 IP 地址；浏览器通过网络与该 IP 地址处的 Web 服务器的__(63)__服务端口间建立一条__(64)__连接；浏览器依照相关协议发送__(65)__命令；Web 服务器根据命令取出文档,发送回来；浏览器释放连接,显示该文档。
(60):A.URL B.EMS C.NDS D.DNS
(62):A.NAT B.EMS C.NDS D.DNS
(63):A.HTML B.HTTP C.SMTP D.SNMP
(64):A.RTP B.IP C.TCP D.UDP
(65):A.TCP B.GET C.UDP D.PUT

MIMD systems can be classified into __(66)__-oriented systems ,high-avai1ability systems and response-oriented systems.The goal of __(66)__ .orlented multiprocess is to obtain h1gh __(66)__ __(67)__ minimal computing cost .The techniques employed by multiprocessor operating systems to achieve this goal take advantage of an inherent processing versus input/output balance in the workload to produce __(68)__ and__(69)__ loading of system __(70)__ .
(66):A.though B.through C.throughout D.throughput
(67):A.at B.of C.on D.to
(68):A.balance B.balanced C.balances D.balancing
(69):A.uniform B.unique C.unit D.united
(70):A resource B.resources C.source D.sources

Soon,more of the informtion we receive via the internet could come __(71)__ in digital wrappers. Wrappers sre made up __(72)__ software code that's targeted to do specific things with the data __(73)__ within them , such as helping to define queries for search engines .They also keep __(74)__ from __(75)__ access to that code.
(71):A.package B.packaged C.packages D.packaging
(72):A.of B.off C.on D.out
(73):A.close B.closed C.enclose D.enclosed
(74):A.insiders B.money C.outsiders D.warehouse
(75):A.gain B.gained C.gains D.gaining

（2001 年度——下午试题）

试题一

阅读下列说明和流程图,回答问题 1 至问题 3,把解答填入答题纸的对应栏内。

[说明]

某计算机厂生产多种型号的计算机,通过分布在世界各地的销售点销售,销售点应在收到计算机商品后的规定时间内把货款汇给该计算机厂。

流程图 1 描述了该厂发货、收款、催款的处理过程。其中商品文件、销售点文件、发货文件和收款文件的记录格式如下:

商品文件 = 商品代号 + 计算机规格名称 + 单价
销售点文件 = 销售点代号 + 销售点名称 + 地址
发货文件 = 发货单号 + 发出日期 + 销售点代号 + 商品代号 + 数量 + 金额
收款文件 = 收款单号 + 收款日期 + 销售点代号 + 商品代号 + 数量 + 金额 + 发货单号
在流程图中,处理 1～处理 3 把当天的发货单合并到发货文件。处理 4～处理 6 把当天的收款单合并到收款文件。每天在处理 3 和处理 5 完成之后,由处理 7 在发货文件中当天已收款的记录上加上已收款标记。处理 8 在月末执行一次,它有三个功能:
①汇总输出本月发货清单;
②删除发货文件中已收款的所有记录,形成一个新的发货文件,作为下月初处理时的初始文件
③产生催款通知单,以便对那些一个月以前已发货但至今仍未收到货款的销售点催款。
[问题 1]
指出流程图 1 中应在哪几个处理框中检查发货单和收款单的错误，并分别说明它们各能指出什么错误。
[问题 2]
如果把流程图 1 中从日收款分类文件到处理 7 的连线改成从日收款文件到处理 7 的连线,则有什么缺点,理由是什么?
[问题 3]
处理 9 每月末执行一次如流程图所示，利用收款文件产生月收款报告并更新收款文件。要求指出它对收款文件做什么更新操作。

试题二
阅读下列说明和流程图,回答问题 1 至问题 3 ,把解答填入答题纸的对应栏内。
[说明]
某考务处理系统具有以下功能:
1.输入报名单;
2.自动编制准考证号;
3.输出准考证;
4.输入成绩清单;
5.输出成绩通知单;
6.输出成绩分布表;
7.输入合格标准、输出录取通知单;
8.试题难度分析,并输出试题难度分析表。
这里给出了实现上述要求的部分不完整的数据流图,其中部分数据流的组成如下所示:
报名单 = 报名号 + 姓名 + 通信地址
考生名册 = 报名号 + 准考证号 + 姓名 + 通信地址
成绩册 = 准考证号 +{ 课程号 + 成绩 } (其中{W}表示 W 重复多次)
准考证 = 报名号 + 姓名 + 准考证号
[问题 1]
指出 0 层图中可以删去的部分。
[问题 2]
在加工 1 子图中将遗漏的数据流添加在答题纸上。
[问题 3]
加工 2 子图分解成如图所示的 4 个子加工及相关的文件(即数据存储)。试在此基础上将相

关的 DFD 成份添加在答题纸上,以完全该加工子图。
[数据流图]
顶层

试题三
在 COMET 型计算机上可以使用试卷上所附的 CASL 汇编语言。阅读程序说明和 CASL 程序,将应填入__(n)__处的字句,写在答题纸的对应栏内。
[程序 3 说明]
子程序 DEHZ 用来对 HZ 编码的字串做解码处理。
HZ 编码是海外华人创造的一种将含有高位为 1 的汉字双字节字串转换成易于在网络中传输的 ASCII 字符串的变换方式。编码过程中,被转换字符串中的原汉字子字符串各字节高位作清零处理,使之成为 ASCII 子字符串,并在其前后两端分别添加 ~{ 和 ~} 作为标记;而对于原 ASCII 子字符串,则将其中的 ~ 改写为 ~~,其余字符不变。
DEHZ 解码子程序则是 HZ 编码的复原过程。复原 ASCII 子字符串过程中遇有 ~~ 字符则改写为一个 ~,遇有 ~{ 则将其后直至 ~} 标记前的各字节高位置 1,复原为汉字子字符串,同时删除其前后标记。~的后续字符不属于以上情况均作为错误处理。
调用该子程序时, GR1 存放原始字符串首地址, GR2 存放还原后的目标字符串首地址。工作寄存器 GR3 用作处理汉字子字符串的识别标志,进入子程序时应初始化为处理 ASCII 子字符串。程序按照 CASL 语言的标准约定,字符串的每个字符只占用一个存储字的低八位。原始字符串和目标字符串均以 0 作为结束标志。
[程序 3]

```
START
DEHZ PUSH 0,GR3
PUSH 0,GR2
PUSH 0,GR1
LEA GR3,0
LOOP __(1)__
CPA GR0,MARK0
JNZ GOON
LEA GR1,1,GR1
LD GR0,0,GR1
CPA GR0,MARK0
__(2)__
CPA GR0,MARK1,GR3
JNZ ERROR
__(3)__
LEA GR1,1,GR1
JMP LOOP
ERROR OUT ERS1R,ERLEN
JMP EXIT
GOON __(4)__
ST GR0,0,GR2
LEA GR2,1,GR2
LEA GR1,1,GR1
```

```
CPA GR0,VO
__(5)__
EXIT P0P GR1
P0P GR2
P0P GR3
RET
V1 DC 1
V0 DC O
DC #0080
MARK0 DC '~ '
MARK1 DC '{ }'
ERSTR DC 'ERROR!'
ERLEN DC 6
END
```

试题四

阅读下列程序说明和 C 代码,将应填入__(n)__处的字句写在答题纸的对应栏内。

[程序 4 说明]

设 M 叉树采用列表法表示,即每棵子树对应一个列表,列表的结构为:子树根结点的值部分(设为一个字符) 和用"( )",括起来的各子树的列表 (如有子树的话) ,各子列表间用",",分隔。例如下面的三叉树可用列表 a(b(c,d),e,f(g,h,i))表示。

本程序输入列表,生成一棵 M 叉树,并由 M 叉树输出列表。假定输入无错误。

[程序 4]

```
#include〈stdio.h〉
#include〈stdlib.h〉
#define M 3
typedef struct node{ char val;
struct node *subTree[M];
} NODE;
char buf[255] ,*str = buf
NODE *d = NULL
NODE *mackTree( ) /*由列表生成 M 叉树*/
{ int k; NODE *s ;
s = __(1)__
s -> val = *str++ ;
for ( k = 0 ; k < M ; k++ ) s-> subTree[k] = NULL ;
if(*str='( '){
k = 0;
do { str++;
s -> subTree[k] = __(2)__ ;
if ( *str == ')' ) { str++; break ; }
k = k+l ;
} while ( __(3)__ );
```

```
}
return s ;
}
void walkTree( NODE *t ) /*由 M 叉树输出列表*/
{ int i ;
if t != NULL ) {
__(4)__
if ( t -> subTree[0] == NULL ) return ;
putchar ( '( ' ) ;
for ( i = 0 ; i < m ; i++) {
__(5)__
if ( i! = M - l && t -> subTree[i+l] != NULL )
putchar ( ', ' ) ;
}
putchar ( ') ' ) ;
}
}

void main( )
{ printf( "Enter exp:" ) ;
scanf( "%S" , str ) ;
d = makeTree() ;
walkTree( d ) ; putchar( '\n') ;
}
```

试题五

阅读下列程序说明和 C 代码,将应填入__(n)__处的字句写在答题纸的对应栏内。

[程序 5 说明]

著名的四色定理指出任何平面区域图均可用四种颜色着色,使相邻区域着不同的颜色。本程序对给定的区域图找出所有可能的不超过四种颜色的着色方案。程序中用 1～4 表示四种颜色。要着色的 N 个区域用 0～N 一 1 编号,区域相邻关系用 adj[][] 矩阵表示,矩阵的 i 行 j 列的元素为 1 ,表示区域 i 与区域 j 相邻；矩阵的 i 行 j 列的元素为 0 ,表示区域 i 与区域 j 不相邻。数组 color[] 用来存储着色结果, color[i] 的值为区域 i 所着颜色。

【程序 5】

```
#include〈stdio.h〉
#define N 10
void output(int color[])/*输出一种着色方案*/
{ int i ;
for ( i = 0 ; i < N ; i++ )
printf( "%4d" , color[i] ) ;
printf( "\n" ) ;
}
int back( int *ip ,int color[] ) /*回溯*/
{ int c = 4 ;
```

```
while ( c == 4 ){
if ( *ip <= 0 ) return 0 ;
--(*ip) ;
c = __(1)__ ;
color[*ip] = -1 ;
}
return c ;
}
/*检查区域 i,对 c 种颜色的可用性*/
int color0k( int i , int c , int[][N] , int color[ ] }
{ int j ;
for ( j = 0 ; j < i ; j++ }
if ( __(2)__ )
return 0 ;
return 1 ;
}

/*为区域 i 选一种可着的颜色*/
int select( int i ,int c ,int adj[][N] , int color[ ] )
{ int k ;
for ( k = c ; k <= 4 ; k++ )
if ( colorOK( __(2)__ ) )
return k ;
return 0 ;
}
int coloring( int adj[][N] ) /*寻找各种着色方案*/
{ int color[N] , i , c , cnt ;
for ( i = 0 ; i < N ; i++ ) color[i] = -1 ;
i = c = 0 ; cnt = 0 ;
while ( 1 ) {
if ( ( c = __(4)__ ) == 0 ){
c = back( &i , color) ;
if ( c == 0) return cnt ;
} else { __(5)__ ; i++ ;
if ( i == N ) {
output(color) ;
++cnt ;
c = back( &i , color ) ;
} e1se c = 0 ;
}
}
}
void main()
{ int adj[N][N] =
```

```
{ {0,1,0,1,1,1,1,1,1,1},
{1,0,1,1,0,1,1,1,1,0},
{0,1,0,1,0,1,1,0,1,1},
{1,1,1,0,1,1,0,0,1,1},
{1,0,0,1,0,1,0,0,0,0},
{1,1,1,1,1,0,1,0,0,1},
{1,1,1,0,0,1,0,0,1,0},
{1,1,0,0,0,0,0,0,1,1},
{1,1,1,1,0,0,1,1,0,1},
{1,0,1,1,0,1,0,1,1,0}
} ;
printf( "共有%d 组解.\n",coloring( adj ) ) ;
}
```

（2001 年度高程答案）

上午答案

(1)D (2)A (3)B (4)A (5)B
(6)D (7)A (8)C (9)B (10)C
(ll)B (12)B (13)D (14)A (15)A
(16)A (17)B (18)B (19)D (20)C
(21)C (22)A (23)D (24)C (25)B
(26)B (27)A (28)B (29)B (30)C
(31)C (32)C (33)A (34)B (35)B
(36)D (37)C (38)A (39)B (40)D
(41)A (42)B (43)B (44)C (45)D
(46)A (47)D (48)D (49)A (50)D
(51)D (52)D (53)B (54)C (55)A
(56)C (57)A (58)C (59)B (60)A
(61)A (62)D (63)B (64)C (65)B
(66)D (67)A (68)B (69)A (70)B
(71)B (72)A (73)D (74)C (75)D

下午答案

试题一
[问题 1)
处理 l 和处理 4：发货单和受款单中的非法销售代号、商品代号；
数量和金额非正整；单位 I*数量，i 金额；输入中的非法字符；
处理 7：重复付款和找不到与收款单位相对应发货单；
[问题 2]
因文件分类，处理时要增加访问外存的时间或需大量内存，从而降低了处理速度

[问题 3]
删除收款文件中的所有记录
试题二
[问题 1]成绩文件可删
试题二

试题三
(1) LD GR0,0GR1
(2) JZE GOON
(3) EOR GR3,V1
(4) OR GR0,V0, GR3
(5) JNZ LOOP
试题四
(1) (NODE*) malloc (sizeof (NODE))
(2) makeTree ( )
(3) *str == ','
(4) putchar (t->val)
(5) walkTree (t->subTree[i])
试题五
(1) color[*ip]
(2) adj[i][j] != 0 && color[j] == c
(3) i,k,adj,color
(4) select (i,c+l,adj,color)
(5) color[i]=c